TITLE OF THE INVENTION

DOUBLE DRAW VIDEO POKER GAMES

BACKGROUND OF THE INVENTION

Field of the Invention:

[01] The present invention is directed to a method, device, and computer readable storage medium for implementing a video poker game which provides a player with two draws.

Description of the Related Art:

[02] Video poker is a popular gambling game found in casinos. What is needed is a new variety of game that some players may prefer.

SUMMARY OF THE INVENTION

[03] It is an aspect of the present invention to provide improvements and innovations in video poker games.

[04] The above aspects can be obtained by a system that includes (a) offering a paytable reflecting an ability of a player to receive two draws, the paytable comprising ranks and respective payouts; (b) dealing a first five card hand to the player; (c) allowing the player to make a first selection comprising any number of cards from the first hand;

(d) replacing cards in the first selection with newly dealt cards to form a second hand; (e) allowing the player to make a second selection comprising any number of cards from the second hand; (f) replacing cards in the second selection with newly dealt cards to form a final hand; (g) determining the rank of the final hand; and (h) paying the final hand according to the rank's respective payout using the paytable.

[05] The above aspects can also be obtained by a system that includes (a) dealing a first five card hand to the player; (b) allowing the player to make a first selection comprising any number of cards from the first hand; (c) replacing the cards from the first selection with newly dealt cards to form a second hand; and (d) if the second hand meets a predefined condition, allowing the player to make a second selection comprising a card or cards from the second hand and replacing the selected card or cards from the second selection with newly dealt cards to form a final hand.

[06] These together with other aspects and advantages which will be subsequently apparent, reside in the details of construction and operation as more fully hereinafter described and claimed, reference being had to the accompanying drawings forming a part hereof, wherein like numerals refer to like parts throughout.

BRIEF DESCRIPTION OF THE DRAWINGS

[07] Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, will become apparent and more readily appreciated from the following description of the preferred embodiments, taken in conjunction with the accompanying drawings of which:

[08] Figure 1 is a flowchart illustrating the operation of a main embodiment of the present invention, according to an embodiment of the present invention;

[09] Figure 2 is a screen shot illustrating a first phase of the present invention, according to an embodiment of the present invention;

[10] Figure 3 is a screen shot illustrating a second phase of the present invention, according to an embodiment of the present invention;

[11] Figure 4 is a screen shot illustrating a third phase of the present invention, according to an embodiment of the present invention;

[12] Figure 5 is a flowchart illustrating a calculation of a return for a paytable for the present invention, according to an embodiment of the present invention;

[13] Figure 6 is a flowchart illustrating a conditional second draw version, according to an embodiment of the present invention;

[14] Figure 8 is a screenshot illustrating a first phase of a multiple hand embodiment, according to an embodiment of the present invention;

[15] Figure 9 is a screenshot illustrating a second phase of a multiple hand embodiment, according to an embodiment of the present invention;

[16] Figure 10 is a screenshot illustrating a third and final phase of a multiple hand embodiment, according to an embodiment of the present invention;

[17] Figure 11 is a screenshot illustrated a third and final phase of another multiple hand embodiment, according to an embodiment of the present invention;

[18] Figure 12 is a screenshot illustrated a second phase of another multiple hand embodiment, according to an embodiment of the present invention;

[19] Figure 13 is a screenshot illustrated a third and final phase of another multiple hand embodiment, according to an embodiment of the present invention; and

[20] Figure 14 is a block diagram illustrating one example of hardware that can be used to implement the present invention, according to an embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[21] Reference will now be made in detail to the presently preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings, wherein like reference numerals refer to like elements throughout.

[22] The present invention relates to video poker games and improvements thereof.

[23] The present invention provides for a video poker game played on a video poker machine allowing the player to take two draws after being dealt an initial first hand. The goal of the game is for the player to make the best hand the player can make. Five cards are dealt to the player from a standard 52 card deck to form a first hand. The player can then choose to discard none or as many of the five cards as the player chooses from the first hand. The video poker machine then deals cards to replace the discarded cards, forming a second hand. The player once again has the option to discard none or as many

of the five cards as the player chooses from the second hand. The video poker machine then deals cards to replace the discarded cards forming a third and final hand.

[24] When the player is dealt the third and final hand, a rank of the hand is determined. A rank of the hand is a category that the hand falls into. For example, the following all can represent ranks of a video poker hand: royal flush, straight flush, four of a kind, full house, straight, three of a kind, two pair, jacks or better, one pair, or any other categorical type of hand.

[25] A paytable is offered at the start of the game which reflects the ability of the player to have two draws. Since the player has the advantage of getting two draws, a standard paytable should be reduced so that the casino can still profit from offering the game. The paytable pays a predetermined amount for each hand rank. A paytable comprises a list of hand ranks and respective payouts.

[26] Typically, only the rank of the third and final hand is determined and paid according to a paytable, while the first and second hands are not paid, with the paytable adjusted accordingly. In another embodiment of the present invention, the rank of the first and/or second and/or third hand can be both determined and paid according to a paytable adjusted for those particular rules. Also typically, the player pays up-front for the ability to get two draws, although less preferred variations could also charge the player for the ability to receive any one or both draws

[27] Figure 1 is a flowchart illustrating the operation of a main embodiment of the present invention, according to an embodiment of the present invention.

[28] The game starts at operation 100, which deals to a player a first five card hand. Before a hand is dealt, any needed initialization process can be performed, such as shuffling a 52 card deck. Also, money for playing the game is collected.

[29] The game then proceeds to operation 102, which allows the player to select from 0-5 cards from the first hand to be discarded. This can be done using any known input method, such as a touch sensitive screen, physical buttons, mouse, etc. Once a player is satisfied with his or her selections, the player can then press a button indicating that the player is finished selecting.

[30] The game then proceeds to operation 104, which then automatically replaces the selected cards from operation 102 with newly dealt cards, forming a second hand.

[31] The game then proceeds to operation 106, which allows the player to select from 0-5 cards from the second hand to be discarded. This selection can be done in the same manner as the first selection in operation 102.

[32] The game then proceeds to operation 108, which then automatically replaces the selected cards from operation 106 with newly dealt cards, forming a third and final hand.

[33] The game then proceeds to operation 110, which then determines the rank of the final hand. The rank can be determined by comparing the cards in the final hand with values for each of the ranks. Sorting the final hand from lowest to highest before comparing can make this process easier to program.

[34] The game then proceeds to operation 112, which pays the player according to the determined rank. The payment is determined using a paytable for that game. If the

player does not have a winning hand, the player is not paid. The game can then return to operation 100 for a new game.

[35] Figure 2 is a screen shot illustrating a first phase of the present invention, according to an embodiment of the present invention.

[36] A paytable 200 is displayed which indicates how much each rank pays depending on how many coins bet. In order to play the game, a player must typically pay up front (either insert cash or use credits) to play a hand of the game. A first hand 202 is also displayed comprising five cards: 3 hearts, ace hearts, 10 diamonds, 7 hearts, and 7 diamonds. Not pictured is a coin output which outputs to the player how many coins/credits the player has.

[37] The player can choose to hold/discard any number and any choices of the initial deal 202. The player can indicate his or her selections by pressing buttons on a video poker machine, touching a screen itself, or using any other input device. In this example the two 7's are held by the player. An indicator such as "HELD" can be displayed alongside the held cards. This corresponds to operation 102 from Figure 1.

[38] Once the player is satisfied with the cards that are held, the player can then select a "Draw 1" button 204 and proceed to the second phase of the game.

[39] Figure 3 is a screen shot illustrating a second phase of the present invention, according to an embodiment of the present invention.

[40] In the second phase of the game, the computer (video poker machine) holds the cards selected to be held and replaces the non-held cards with new cards (see operation

104 from Figure 1). A second hand 300 is displayed which comprises the held cards (7 hearts, 7 diamonds) as well as new cards which replaced the non-held cards (10 hearts, 8 diamonds, 2 hearts).

[41] An optional discard output 302 can be displayed which displays the cards which were discarded from the first phase of the game. These cards are displayed so that the player can take into consideration which cards were discarded when choosing which cards to hold/discard for the second draw. For example, if the player has four to a royal flush on the second hand, but one of the discarded cards is the card the player needs to complete the royal flush, then the player should not discard only one card to try for the royal flush since it would be impossible to make.

[42] The player then makes a second selection of cards he or she wishes to hold/discard from the second hand in the same manner as before, See operation 106 from Figure 1. In this example the payer keeps the two 7's and discards the other 3 cards.

[43] When the player wishes to proceed to the final phase of the game, the player can press the "Draw 2" button 304.

[44] Figure 4 is a screen shot illustrating a third phase of the present invention, according to an embodiment of the present invention.

[45] The computer (video poker machine) holds the cards selected to be held and replaces the non-held cards with new cards. A third (final) hand 400 is dealt/displayed which comprises the held cards from phase 2 as well as new cards which replaced the held cards. This corresponds to operation 108, from Figure 1.

[46] The third hand 400 is the final hand, and the game is now over. The rank of the hand is determined (see operation 110, from Figure 1), and if the player has a winning hand the paytable can have an indicator 402 highlighting the rank of the paying hand, and also the amount of money won (not pictured). In this case, the rank of the hand is 3 of a kind. If the player has a winning hand at this phase, the player is paid (see operation 112 from Figure 1). The player can then optionally begin a new game.

[47] It is noted that typically the same 52 card deck is used for all phases of each game, and is typically shuffled at the beginning. Other non-standard decks can be used with the game as well.

[48] In order to make the game described above viable for a casino, a paytable must be computed which has an optimal return in an acceptable casino range. The optimal return can be defined as the best possible return for a player that plays a game perfectly. A typical acceptable casino range would comprise a return between 94% and 104%. The return should be set high enough to encourage players to play, but low enough so that the casino will still make a profit. Note that games that return over 100% can still return a profit to the casino if all players do not follow the optimum strategy

[49] Figure 5 is a flowchart illustrating a calculation of a return for a paytable for the present invention, according to an embodiment of the present invention.

[50] A paytable can be designed by a game designer to choose the payouts the designer wishes. Of course, the paytable should result in an acceptable optimal return. A calculation is performed to determine the optimal return of the paytable.

[51] The calculation starts with operation 500, which cycles through all initial 5 card hands (2,598,960 - which is computed as 52!/(47!*5!).

[52] The calculation then proceeds to operation 502, which cycles through all 32 ways to play the initial hand. There are 32 ways to play the initial hand because each of the five cards can be held or discarded, so 2^5=32.

[53] The calculation then proceeds to operation 504, which replaces the discarded cards from each operation 502 in all possible ways to create a second hand. If all five cards are discarded, then there are 1,533,939 ways to replace the discards 47!/(42!*5!). If four cards are discarded, then there are 47!/(43!*4!) ways to replace the discards, etc.

[54] The calculation then proceeds to operation 506, which cycles through all 32 ways to play the second hand.

[55] The calculation then proceeds to operation 508, which replaces the discarded cards from each operation 506 in all possible ways to create a third and final hand.

[56] The calculation then proceeds to operation 510, which determines (and stores) a payout for the final hand depending on the rank of the hand and the paytable being used.

[57] When operations 502-510 are completed, the calculation proceeds to operation 512, which stores the best payout of all of the ways to play the particular initial hand is saved. The calculation then returns to operation 500 to cycle through the next dealt hand.

[58] When all initial hands are cycled through in operation 500 (invoking operations 502-512), the calculation then proceeds to operation 514 which determines the optimal

payout of the paytable. This is determined by summing all of the saved best payouts from operation 512 and dividing by the number of initial hands.

[59] If the optimal return of the paytable is not at a level acceptable to the game designer, the designer can continuously adjust payouts of the table and perform the above described method again.

[60] The above described calculation, when the player discards all 5 cards initially and all five cards again, requires a large number of hands to be dealt.

[61] A number of optional shortcuts can be implemented in order to cut down on the processing time.

[62] First, although there are 2,598,960 ways to choose the initial 5 cards out of a 52-card deck there are only 134,459 classes of hands. Analyzing each class and weighting by the number of similar classes produces the same results as analyzing all 2,598,960 hands. For example if the player has four kings and a queen it does not make any difference what suit the queen is. Rather than analyzing four hands (one for each suit of the queen) we can pick an arbitrary suit for the queen and weight the results by 4.

[63] Second, if it can be shown that the player should never discard all five cards on the initial deal then this option can be ignored. This is the most time consuming of the 32 ways to play the hand due to the large number of ways to choose 5 replacement cards out of 47.

[64] Third, holding one card and discarding four on the initial deal is still time consuming. However depending on the card kept the expected value can be predicted

within a small range, varying only because of which cards are discarded. A time saving measure that can be implemented is to determine an average value for keeping each of the 13 ranks on the initial deal. This could be determined by cycling through all 495 (12!/(8!*4!)) ways of choosing 4 ranks out of 12 for the discarded cards, assuming it can be shown the player should never hold a singleton over a pair. Arbitrary suits can be assigned, so that none of the discarded suits match the card held. This shortcut will admittedly not produce perfect results but the degree of error should be very small.

[65] Also, another shortcut can be utilized which skips the processing of obviously bad plays. For example, the program only holds two cards if they are any of the following: (1) a pair, (2) suited cards, (3) consecutive cards. A player should typically not hold 2 cards if one of these conditions were not met.

[66] Table I is an example of an optimal paytable computed using the above described methods. A player would have utilized optimal strategy on both draws in order to achieve these optimal results.

Table I

Hand	*Prob*	*Pays*	*Return*
Nonpaying hand	0.594120	0	0.000000
Two pair	0.185606	1	0.185606
Three of a kind	0.117042	2	0.234084
Straight	0.026194	3	0.078583
Flush	0.038588	3	0.115764
Full House	0.030974	5	0.154872
Four of a kind	0.006867	20	0.137333
Straight Flush	0.000478	50	0.023902
Royal Flush	0.000130	500	0.065058
Total	1.000000		0.995202

[67] The probability column is the probability of that hands resulting after two draws if the player uses optimal strategy. This can be determined by tabulating the ranks of the final best hands in operation 510 from Figure 5, and dividing by the total number of hands. The return column represents the percentage of the final return that the respective rank contributes to the overall return. This can be determined for each rank by multiplying the payout column by the respective payout column.

[68] Note the paytable in Table I has a final optimal return of .995202 (99.5202%). This is obtained by summing all of the returns for the individual ranks. The return of 99.5% is an ideal return for this game, as it is high enough to attract players, but a casino will still make a profit, especially since most players will not be utilizing optimal strategy.

[69] The results in Table I were created using shortcuts as described above. However, due to the shortcuts taken, the accuracy of the results in Table I contains an X margin of error within 0.1%.

[70] Appendix A contains one example of code written to implement the above method, written in C++. Of course, most other programming languages can be used to implement the method.

[71] In another embodiment of the present invention, a conditional second draw version of the present invention can be implemented. In this embodiment, a second draw can invoked upon certain conditions, but otherwise is not offered to the player. For example, a one draw game can be offered, but if the player achieves a "four to a royal" hand after the first draw, the game automatically offers the player a second draw. In this

way, the player has an additional try to make a royal flush. For the second draw, the player can either be limited to discarding a limited number of cards (such as 1, 2, 3, 4 or 5) or can discard any amount of cards (1-5) the player wishes. In a less preferred embodiment, for the first draw the player can either be limited to discarding a limited number of cards (such as 1, 2, 3, 4 or 5) or can discard any amount of cards (1-5) the player wishes.

[72] Figure 6 is a flowchart illustrating a conditional second draw version, according to an embodiment of the present invention.

[73] The game starts at operation 600, which deals to a player a first five card hand. Before a hand is dealt, any initialization can be performed, such as shuffling a 52 card deck.

[74] The game then proceeds to operation 602, which allows the player to select from 0-5 cards from the first hand to be discarded.

[75] The game then proceeds to operation 604, which then replaces the selected cards from operation 200 with newly dealt cards, forming a second hand.

[76] The game then proceeds to operation 606, which checks the second hand for a predetermined condition(s). The predetermined criteria typically a type of hand, regardless of whether it would comprise a paying hand or not. For example, one predetermined criteria could whether the hand comprises a "four to a royal" hand. A four to a royal hand is where the player has four cards which can comprise a royal flush, but a fifth card to complete the royal flush is needed. If the check returns a negative result (the

hand does not meet the predefined criteria), then the second hand is considered the "final hand" and the game proceeds to operation 610.

[77] If the check in operation 606 determines that the predetermined condition(s) is met, then the game then proceeds to operation 608, which allows the player to select card(s) from the second hand to be discarded. The player may be limited to selecting a fixed number of cards (such as one card), or the player may be allowed to select any amount of cards to select the player wishes. Once those card(s) are selected, then they are discarded and replaced with newly drawn card(s) to comprise a final hand.

[78] From operations 606 or 608, the game proceeds to operation 610, which determines a rank of the final hand. The rank can be determined by comparing the cards in the final hand with values for each of the ranks. Sorting the final hand from lowest to highest before comparing can make this process easier to program.

[79] The game then proceeds to operation 612, which pays the player according to the determined rank. Of course, if the player does not have a winning hand, the player is not paid. The game can then return to operation 600 for a new game.

[80] The paytable in this embodiment should be adjusted to compensate for the additional player advantage of a second draw in certain circumstances. Figure 7 is a flowchart illustrating a method of computing such an adjusted paytable, according to an embodiment of the present invention.

[81] In operation 700, the method deals initial hands. The number of ways to arrange the initial 5 cards out of a 52-card deck is 2,598,960. However in video poker many

initial hands have exactly the same value and possible outcomes. For example if the initial hand were four queens and a king it would not make any difference what suit the king was. So to speed analysis it would be fair to only analyze this hand once, giving the king an arbitrary suit, and weighting the possible outcomes by 4 (one for each possible suit of the king). Combining all similar hands the number of initial hands necessary to evaluate is only 134,459.

[82] After initial hands are dealt in operation 700, the operation proceeds to operation 702 which cycles through each way to play each dealt initial hand. After the initial hand is dealt the player may choose to keep or discard each card. Therefore there are $2^5=32$ possible ways to play each hand. The number of possible replacement cards depends on the number of cards kept. Table I shows the number of ways to choose each number of discards out of the initial five cards and the total combinations of replacement cards. The product column shows the product of these two numbers.

Table I

Discards	Sets	Draw Combinations	Product
0	1	1	1
1	5	47	235
2	10	1081	10810
3	10	16215	162150
4	5	178365	891825
5	1	1533939	1533939
Total	32		2598960

[83] The lower right cell shows the number of possible combinations on the draw is also conveniently 2,598,960. In other words given any initial hand all final hands are still possible. A slow method of analyzing standard video poker would be to score all 2598960 possible hands on the draw. Even with the shortcut limiting the initial hands to

134,459, analyzing all 2598960 draw hands for each one would necessitate scoring over 349 billion hands.

[84] To speed up processing time, the method can only score each possible hand once. It takes only a few seconds to score 2,598,960 hands. Each hand can be numbered sequentially and the hand value is stored in an array. Likewise it is determined if each possible hand contains four to a royal flush and another array of 2,598,960 elements is used to store whether each hand contains exactly four to a royal.

[85] For each of the 134,459 initial hands the computer proceeds to operation 704 which loops through all 2,598,960 possible hands on the draw. For each one the method checks to see which cards are common to both the initial hand and the final hand. According to which cards are common to both the method will increment the value of a two dimensional array, one dimension for each way to play the initial hand and another for each possible outcome of the final hand. To be more specific each card position is given an indicator of a power of 2 (1, 2, 4, 8, and 16). If an initial card is part of the final hand a counter is incremented with the corresponding power of 2 for that card position.

[86] In addition, as the program cycles through all 2,598,960 final hands it flag hands that meet a predefined condition, in this case having four to a royal flush, in operation 706. In these cases the hand is sent to another subroutine which evaluates the possible outcomes of a second draw. These possible outcomes are then stored in another two dimensional array for the possible outcomes of the second draw. The program is careful to consider which cards have already been removed from the deck, either from the initial deal or the first draw.

[87] After looping through all 2,598,960 possible final hands, the method then proceeds to operation 708 which calculates the expected value of each possible play. This is done by weighting each payoff and its probability for all 32 ways to play each hand. In addition the expected value of the second draw is added to each possible play. The play with the greatest expected value is selected and the possible outcomes stored in an array for the entire game.

[88] The possible draw combinations should be weighted inversely so that each possible play has the same weight. For example if the player discards all five initial cards there are 1,533,939 ways to get 5 out of 47 replacements cards. However if the player only discards one card there are 47 ways to get a replacement card. So the draw combinations are weighted proportionally to the inverse of their total.

[89] When all the initial hands are cycled through, the method proceeds to operation 710 which displays the total combinations for the entire game. Given the frequency of each final hand, both on the first and second draw, the total return can be easily calculated.

[90] Table II shows the results for the first draw in a Jacks or Better game.

Table II

Hand	Pays	Combinations	Probability	Return
Nothing	0	919,136,157,346,512	0.548937	0.000000
Jacks or Better	1	351,887,245,651,872	0.210158	0.210158
Two pair	2	215,333,068,522,320	0.128604	0.257208
Three of a kind	3	124,087,393,826,928	0.074109	0.222327
Straight	4	18,008,776,947,168	0.010755	0.043022
Flush	5	17,847,429,833,520	0.010659	0.053295
Full House	6	19,161,504,706,176	0.011444	0.068663
Four of a kind	20	3,937,460,675,904	0.002352	0.047032
Straight Flush	50	168,076,417,824	0.000100	0.005019

Royal Flush	800	52,636,568,544	0.000031	0.025149
No Hand	0	4,771,612,948,032	0.002850	0.000000
Total		1,674,391,363,444,800	1.000000	0.931872

[91] The "No Hand" column means the player had 4 to a royal and accepted the option for a second draw. The probability column shows this happens 0.285% of the time, or once every 351 hands. The return on the first draw is 93.19%.

[92] Table III shows the results on the second draw. Note that 99.715% of the time the player will not get a second draw, because he didn't have four to a royal on the first draw. The lower right cell shows the second draw contributes 5.53% to the game return.

Table III

Hand	Pays	Combinations	Probability	Return
Nothing	0	2,543,373,927,048	0.001519	0.000000
Jacks or Better	1	967,907,545,944	0.000578	0.000578
Two pair	2	0	0.000000	0.000000
Three of a kind	3	0	0.000000	0.000000
Straight	4	354,825,920,376	0.000212	0.000848
Flush	5	778,947,141,588	0.000465	0.002326
Full House	6	0	0.000000	0.000000
Four of a kind	20	0	0.000000	0.000000
Straight Flush	50	19,827,215,004	0.000012	0.000592
Royal Flush	800	106,731,198,072	0.000064	0.050995
No Hand	0	1,669,619,529,402,040	0.997150	0.000000
Total		1,674,391,142,350,070	1.000000	0.055338

[93] Table IV shows the final outcome of the player hand and the total return of the game of 98.72%.

Table IV

Hand	Pays	Combinations	Probability	Return
Nothing	0	921,679,531,273,560	0.550456	0.000000
Jacks or better	1	352,855,153,197,816	0.210736	0.210736
Two pair	2	215,333,068,522,320	0.128604	0.257208
Three of a kind	3	124,087,393,826,928	0.074109	0.222327
Straight	4	18,363,602,867,544	0.010967	0.043869

Flush	5	18,626,376,975,108	0.011124	0.055621
Full House	6	19,161,504,706,176	0.011444	0.068663
Four of a kind	20	3,937,460,675,904	0.002352	0.047032
Straight Flush	50	187,903,632,828	0.000112	0.005611
Royal Flush	800	159,367,766,616	0.000095	0.076144
Total		1,674,391,363,444,800	1.000000	0.987211

[94] Table V compares the probability of each hand in regular video poker with this embodiment (offering a second draw on a four to a royal hand), both based on their respective optimal strategy for the given pay table. Note the greatly increased probability of getting a royal flush. In this paytable this game results in 3.82 times as many royals as regular video poker.

Table V

Hand	Regular	Second draw on 4 to a royal	Regular	Second draw on 4 to a royal
Nothing	0.545080	0.550456	1 in 1.83	1 in 1.82
Jacks or better	0.215040	0.210736	1 in 4.65	1 in 4.75
Two pair	0.129249	0.128604	1 in 7.74	1 in 7.78
Three of a kind	0.074428	0.074109	1 in 13.44	1 in 13.49
Straight	0.011284	0.010967	1 in 88.62	1 in 91.18
Flush	0.010913	0.011124	1 in 91.63	1 in 89.89
Full House	0.011511	0.011444	1 in 86.88	1 in 87.38
Four of a kind	0.002362	0.002352	1 in 423.35	1 in 425.25
Straight Flush	0.000108	0.000112	1 in 9250.88	1 in 8910.9
Royal Flush	0.000025	0.000095	1 in 40173.19	1 in 10506.46

[95] Appendix B contains one example of code written to implement the above method, written in C++. Of course, most other programming languages can be used to implement the method.

[96] It is helpful to compare the methods illustrated in Figures 5 and 7 (and the code in Appendix A and B, respectively). The first method (illustrated in Figure 5, the

accompanying description and code attached in Appendix A) uses shortcuts (described above) to make the processing time more manageable. The shortcuts come at the expense of a very slight decrease in accuracy.

[97] The second method (illustrated in Figure 7, the accompanying description, and the code attached in Appendix B) uses an exact method by cycling through every possible hand with no shortcuts taken that decrease accuracy.

[98] Some games, such as the basic double draw game, are betted suited for the first method of analysis because the second will take too long. On the other hand, the conditional double draw game contains less combinations to deal and can be analyzed by the second, more exact method.

[99] In addition to using a four to a royal hand as the triggering condition for a second draw, any other hand can be used as well. For example, a hand comprising four to a straight flush and/or four to a royal can be used. A hand comprising three of a kind can also be used. A hand that comprises all nonpaying hands can also be used (i.e. if after the first draw a player does not have a paying hand, the player can then be given a second draw). Any paying or nonpaying hand, and any combinations particular paying and/or nonpaying hands can be used as the triggering criteria/criterion. Paytables for these games can be generated using the methods described above.

[100] In a further embodiment of the present invention, multiple hands can be played simultaneously. In this manner, players can enjoy faster play. Playing multiple hands can be accomplished at least three ways.

[101] Figure 8 is a screenshot illustrating a first phase of a multiple hand embodiment, according to an embodiment of the present invention.

[102] A bottom first hand 800 is dealt, with the cards: 3 spades, 5 diamonds, 10 clubs, 4 spades, and 10 spades. Assume a player holds the 10 clubs and the 10 spades, and then draws.

[103] Figure 9 is a screenshot illustrating a second phase of a multiple hand embodiment, according to an embodiment of the present invention.

[104] The game then deals three (or any number can be used) rows of hands, while keeping the selected discards (10 clubs and 10 spades). Thus, there is a bottom second hand 800, a middle second hand 802, and a top second hand 804. Each of these three hands maintains the cards that were selected to be held in the first phase. Also illustrated are three draw buttons, a bottom draw button 806, a middle draw button 808, and a top draw button 810.

[105] A player selects his or her discards from each row of cards. The player then can push each of the three draw buttons in order to receive replacement cards for the selected cards for each row. In the example illustrated in Figure 9, the bottom hand comprises: 7 clubs, 2 spades, 10 clubs, 7 hears, 10 spades. The player decides to keep the 7 clubs, 10 clubs, 7 hearts, and 10 spades, and then presses the bottom draw button 806.

[106] Figure 10 is a screenshot illustrating a third and final phase of a multiple hand embodiment, according to an embodiment of the present invention.

[107] A bottom final hand 1000 is completed by redealing cards that were previously not selected to be held. The bottom final hand 1000 comprises: 7 clubs, kind spades, 10 clubs, 7 hears, and 10 clubs, with a rank of 2 pair. While not pictured, the middle hand and the top hand can be completed in the same manner.

[108] Figure 11 is a screenshot illustrated a third and final phase of another multiple hand embodiment, according to an embodiment of the present invention.

[109] Figure 11 follows Figures 8 and 9. Instead of dealing one hand for each row after the final selection as illustrated in Figure 10, the game can deal multiple hands (for example 3, although any number can be used) for the final hand, while keeping the selected cards. In the case of Figure 10, 3 hands are dealt after the second draw for the bottom hand. Each hand is ranked and paid appropriated.

[110] Figure 12 is a screenshot illustrated a second phase of another multiple hand embodiment, according to an embodiment of the present invention.

[111] After the operations as described for Figure 8, the game can deal a second bottom hand 1200, but not deal a middle or top hand. The player then selects the cards to keep from the bottom second hand 1200. In this case the player decides to keep the 7 clubs, 10 clubs, 8 hearts and 10 spades, while discarding the 2 spades. The player presses the "draw 2" button 1202 to proceed to the third and final phase of the game.

[112] Figure 13 is a screenshot illustrated a third and final phase of another multiple hand embodiment, according to an embodiment of the present invention.

[113] After the player has made his or her second selection of discards (as described in the accompanying description to Figure 12), the game then deals final hands (any number of final hands can be used).

[114] Figure 13 illustrates a bottom final hand 1300, a middle final hand 1302, and a top final hand 1304. The hands all comprise cards kept from the second selection, while replacing any card(s) that were not selected to be held.

[115] The multiple hand embodiments described above can be applied to any of the games described herein. It is also noted that each "line" of the multiple hand embodiments (i.e. first hand, second hand, final hand) is dealt from its own deck.

[116] In yet a further variation of the present invention, a standard 52 card deck can be used with one additional (or multiple) "double draw card(s)." A player by default will receive one draw after the initial deal. However, if the player is dealt the "double draw card" (either on the initial deal or on the first draw), then the player will be entitled to a second draw. The double draw card should be indicated in any way, such as being shown but then replaced by the next card in the deck so game play proceeds normally. Alternatively, instead of using a double draw card, the double draw can be triggered at random using a predetermined probability.

[117] The double draw card variation of the game can be analyzed by using a combination of the first calculation method described above and a method for analyzing a single draw video poker game (by running through all possible initial hands, draw combinations, and then taking the best possible hand to be made for each initial hand). If the player receives a double draw card on the deal the first method described would be

used. Otherwise if the player did not get a double draw card on the deal the program would calculate the return using the method just described above for single draw video poker. A weighted average of the results of both methods would be taken according to the probability of getting a second draw card on the first draw. This method applies to a game where the double draw card can come out on the initial deal only. In order to analyze a game where the double draw card could come out on the first draw as well, then if the player did not get a double draw card on the initial deal then the expected value should be determined using both the double draw method and that of single draw video poker. A weighted average should be taken according to the probability of getting a double draw card on the first draw of each play on the deal, with the highest expected value taken.

[118] It is further noted that any of the games described herein can be played with any kind of deck, either standard or nonstandard. Wildcards can also be used. Analysis of payouts on any variation of the game can be achieved using the methods described herein.

[119] Figure 14 is a block diagram illustrating one example of hardware that can be used to implement the present invention, according to an embodiment of the present invention. Typically, an electronic gaming device (EGD) is used to implement the present invention.

[120] A processing unit 1400 is connected to a ROM 1402, RAM 1404, and a storage unit 1406 such as a hard drive, CD-ROM, etc. The processing unit 1400 is also connected to an input device(s) 1408 such as a touch sensitive display, buttons, keyboard,

mouse, etc. The processing unit 1400 is also connected to an output device(s) 1410 such as a video display, audio output devices, etc. The processing unit 1400 is also connected to a financial apparatus 1412, which can accept payments and handle all facets of financial transactions. The processing unit 1400 is also connected to a communications link 1414 which connects the gaming device to a casino network or other communications network.

[121] It is also noted that any and/or all of the above embodiments, configurations, variations of the present invention described above can mixed and matched and used in any combination with one another. Any claim herein can be combined with any others (unless the results are nonsensical). Further, any mathematical formula given above also includes its mathematical equivalents, and also variations thereof such as multiplying any of the individual terms of a formula by a constant(s) or other variable.

[122] Moreover, any description of a component or embodiment herein also includes hardware, software, and configurations which already exist in the prior art and may be necessary to the operation of such component(s) or embodiment(s).

[123] The many features and advantages of the invention are apparent from the detailed specification and, thus, it is intended by the appended claims to cover all such features and advantages of the invention that fall within the true spirit and scope of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact

construction and operation illustrated and described, and accordingly all suitable modifications and equivalents may be resorted to, falling within the scope of the invention.

APPENDIX A

```
//
//

#include <iostream.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <stdio.h>

struct card
{
  int r;
  int s;
  int in;
};

void EnterAnyHand(void);

void gamereturn(void);
void dealhand(card deal[], card deck[]);
void TestHands();
void presetup1();
void presetup2();
void presetup3();
void presetup4();
void presetup5();
void presetup6();
void setsuit(int a, int b, int c, int d, int e, card player[], int w);
void setrank(int a, int b, int c, int d, int e, card player[]);
void playhand1(card deal[], int w);
void entercards(card deal[],int numcards);
void sort(card t[]);
int score(card t[]);
int samenum(int c1, int c2, int c3);
int samenum(int c1, int c2, int c3, int c4);
void fast0(int total[]);
void fast1(int r, int s, int total[]);
void fast2(card t[], int total[]);
void fast3(card t[], int total[]);
void slow1(card deal[], card deck[], int total[]);
void slow2(card deal[], card deck[], int total[]);
void slow4(card deal[], card deck[], int total[]);
void slow5(card t[], int total[]);
double advicesubmit(int keep, card inhand[], card deck[], int total[]);
double advicesetup(card deal[], card disc[], int total[]);
double advicesubmit2(int keep, card inhand[], card deck[], int count,
int total[], int RemCards);
int strflushrank(int r1, int r2, int r3);
void paytable(void);
void seepaytable(void);
void Draw4Cards(void);
void setrank(int a, int b, int c, int d, int e, card player[]);
```

```
int abs(int x);
int __fastcall RandNum();


#define _WIN32_WINNT 0x0400

#include <windows.h>
#include <wincrypt.h>

char * pokerhands[]={
      "Nonpaying hand     ", // 0
      "Pair               ",
      "Two pair           ",
      "Three of a kind    ",
      "Straight           ",
      "Flush              ",
      "Full House         ",
      "Four 5-K           ",
      "Four 2-4           ",
      "Four A             ",
      "Straight Flush     ",
      "Royal Flush        ",
      "N/A                ", // 12
      "N/A                ", // 13
      "Total              "}; // 14

int indeck[14][5];
int pay[14]={0,0,1,2,3,3,5,20,20,20,50,500,0,0};
double gtotal[15];
int
numhand,FinHand,totalw,bestplay,curtime,begtime,details,Draw2Combin[6];
double TotRet=0;
bool InitPair;

unsigned __int64
Draw5Tot[15],Draw4Tot[15],Draw3Tot[15],Draw2Tot[15],Draw1Tot[15],Draw0T
ot[15],Draw5Count,Draw4Count,Draw3Count,Draw2Count,Draw1Count,Draw0Coun
t;
/*
double Draw4CardsProb[13][15]={
176952,     18196.8,    28373.3,    15792,      3195.07,    3992.01,
      2899.73,    169.116,    306.906,    0.977466,   20.7718,
      0.977107,   0,    0,    249899.658373,
176825,     17914.9,    28166.9,    15683.1,    3951.74,    3965.61,
      2890.27,    167.687,    306.319,    0.977466,   26.3868,
      0.977107,   0,    0,    249899.867373,
176715,     17578.9,    27949.7,    15569.9,    4807.02,    3893.36,
      2880.29,    166.041,    305.924,    0.977466,   31.5261,
      0.977107,   0,    0,    249899.615673,
176717,     17121.9,    27736.6,    15456.1,    5671.67,    3817.51,
      2870.39,    416.086,    53.7378,    0.977466,   36.5624,
      0.977107,   0,    0,    249899.510773,
176509,     17172.1,    27744.2,    15404.5,    5875.76,    3820.83,
      2865.41,    414.572,    53.7039,    0.977466,   38.1979,
      0.977107,   0,    0,    249900.228373,
```

```
176379,     17350.6,    27779.4,    15422.1,    5736.31,    3856.38,
      2867.42,    414.429,    54.161,     0.977466,   38.3636,
      0.977107,   0,    0,    249900.118173,
175841,     17893.8,    27779.6,    15422.2,    5715.8,     3870.86,
      2867.42,    414.271,    54.3192,    0.977466,   38.3799,
      0.977107,   0,    0,    249899.604673,
174928,     18768.8,    27744.5,    15404.7,    5823.36,    3857.13,
      2865.41,    414.02,     54.2561,    0.977466,   38.2305,
      0.977107,   0,    0,    249900.361173,
175115,     18937.5,    27625.8,    15378.1,    5581.94,    3891.26,
      2862.02,    413.291,    54.3081,    0.977466,   32.7621,
      6.87716,    0,    0,    249899.835826,
151758,     42745.2,    27838.9,    15491.9,    4735.63,    3953.38,
      2871.92,    415.235,    54.5039,    0.977466,   27.726,
      6.87716,    0,    0,    249900.249526,
152662,     42278.4,    28056,      15605.1,    3908.76,    4005.14,
      2881.91,    417.185,    54.5978,    0.977466,   22.5696,
      6.87716,    0,    0,    249899.517026,
153969,     41380.6,    28262.3,    15713.9,    3146.83,    4037.6,
      2891.36,    419.127,    54.6714,    0.977466,   16.9385,
      6.87716,    0,    0,    249900.181526,
161421,     38071.1,    27925.4,    11975.5,    3450.77,    4601.22,
      2150.53,    161.402,    54.3456,    61.6466,    17.4291,
      9.80623,    0,    0,    249900.14953}; */

double Draw4CardsEV[13]={
0.514447850687089,
0.521959679977702,
0.529467066600741,
0.53693986264921,
0.53621042746518,
0.536096649052952,
0.536007938825956,
0.535992969413584,
0.552202828208593,
0.544692831418147,
0.537168815295431,
0.529815078403588,
0.528281433949789};

int weight42[6]={10,76,741,9880,202540,8506680};
int weight43[6]={5,39,390,5330,111930,4812990};
int weight44[6]={1,8,82,1148,24682,1086008};
int weight45[6]={10,82,861,12341,271502,12217590};
int weight46[6]={10,84,903,13244,297990,13707540};
int weight47[6]={5,43,473,7095,163185,7669695};


int main()
{
      int ch;
      do
      {
            cerr << "1.  return of game\n";
            cerr << "2.  change pay table\n";
            cerr << "3.  see pay table\n";
            cerr << "4.  enter any hand\n";
```

```
            cerr << "5.  draw 4 card probabilities\n";
            cerr << "9.  quit\n";
            cin >> ch;
            if (ch==1)
                  gamereturn();
            else if (ch==2)
                  paytable();
            else if (ch==3)
                  seepaytable();
            else if (ch==4)
                  EnterAnyHand();
            else if (ch==5)
                  Draw4Cards();
      }
      while (ch!=9);
      return 1;
}

void EnterAnyHand(void)
{
      int i;
      card player[5];
      details=1;
      InitPair=true;
      for (i=0; i<=15; i++)
            gtotal[i]=0;
      entercards(player,5);
      sort(player);
      playhand1(player,1);
      gtotal[14]=0;
      for (i=13; i>=0; i--)
            gtotal[14]+=gtotal[i];
      for (i=13; i>=0; i--)

      printf("%s\t%i\t%f\t%f\n",pokerhands[i],pay[i],gtotal[i],gtotal[i
]/gtotal[14]);
}

void paytable(void)
{
      for (int i=1; i<=13; i++)
      {
            cerr << "Enter pay for a " << pokerhands[i] << "(current
pay is " << pay [i] << "):  ";
            cin >> pay[i];
      }
}

void seepaytable(void)
{
      for (int i=13; i>=1; i--)
            cerr << pokerhands[i] << "\t" << pay[i] << "\n";
}


void gamereturn(void)
{
```

```
        int i;
        for (i=0; i<=14; i++)
        {
              gtotal[i]=0;
              Draw5Tot[i]=0;
              Draw4Tot[i]=0;
              Draw3Tot[i]=0;
              Draw2Tot[i]=0;
              Draw1Tot[i]=0;
              Draw0Tot[i]=0;
        }
        Draw5Count=0;
        Draw4Count=0;
        Draw3Count=0;
        Draw2Count=0;
        Draw1Count=0;
        Draw0Count=0;
        details=0;
        TotRet=0;
        numhand=0;
        FinHand=0;
        totalw=0;
        begtime=time(NULL);
//      TestHands();
        presetup1();
        presetup2();
        presetup3();
        presetup4();
        presetup5();
        presetup6();
        printf("Total Return=\t%f\n",TotRet);
        printf("total hands =\t%i\n",numhand);
        printf("total weight =\t%i\n",totalw);
        printf("hands calculated = \t%i\n",FinHand);
        printf("gave ev =\t%f\n",TotRet/(double)totalw);
/*      for (i=13; i>=0; i--)
        {
              gtotal[i]/=(double)totalw;
              printf("%s\t%i\t%f\n",pokerhands[i],pay[i],gtotal[i]);
        } */
        printf("\nKeep 5 Cards\n");
        for (i=0; i<=14; i++)
              printf("%s\t%i\t%I64i\n",pokerhands[i],pay[i],Draw0Tot[i]);
        printf("Total hands=\t%I64i\n",Draw0Count);

        printf("\nKeep 4 Cards\n");
        for (i=0; i<=14; i++)
              printf("%s\t%i\t%I64i\n",pokerhands[i],pay[i],Draw1Tot[i]);
        printf("Total hands=\t%I64i\n",Draw1Count);

        printf("\nKeep 3 Cards\n");
        for (i=0; i<=14; i++)
              printf("%s\t%i\t%I64i\n",pokerhands[i],pay[i],Draw2Tot[i]);
        printf("Total hands=\t%I64i\n",Draw2Count);

        printf("\nKeep 2 Cards\n");
        for (i=0; i<=14; i++)
```

```
            printf("%s\t%i\t%I64i\n",pokerhands[i],pay[i],Draw3Tot[i]);
      printf("Total hands=\t%I64i\n",Draw3Count);

      printf("\nKeep 1 Cards\n");
      for (i=0; i<=12; i++)

      printf("card=\t%i\t%I64i\t%f\n",i,Draw4Tot[i],Draw4CardsEV[i]);
      printf("Total hands=\t%I64i\n",Draw4Count);

      printf("\nKeep 0 Cards\n");
      for (i=0; i<=14; i++)
            printf("%s\t%i\t%I64i\n",pokerhands[i],pay[i],Draw5Tot[i]);
      printf("Total hands=\t%I64i\n",Draw5Count);
}




void dealhand(card deal[], card deck[])
{
      int i,j,rn,count;
      count=0;
      for (i=0; i<=12; i++)
      {
            for (j=0; j<=3; j++)
            {
                  deck[count].r=i;
                  deck[count].s=j;
                  deck[count].in=1;
                  count++;
            }
      }
      for (i=0; i<=4; i++)
      {
            do
            {
                  rn=abs(RandNum()%52);
            }
            while (deck[rn].in==0);
            deck[rn].in=0;
            deal[i].r=deck[rn].r;
            deal[i].s=deck[rn].s;
      }
}

void entercards(card deal[],int numcards)
{
      int i,j;

      char entcard[2];
      char
charrank[18]={'2','3','4','5','6','7','8','9','t','j','q','k',
        'a','T','J','Q','K','A'};
      char charsuit[8]={'h','d','c','s','H','D','C','S'};
      int intrank[18]={0,1,2,3,4,5,6,7,8,9,10,11,12,8,9,10,11,12};
      int intsuit[8]={0,1,2,3,0,1,2,3};
```

```
    for (i=0; i<numcards; i++)
    {
          cerr << "Enter your card #" << i << ":  ";
          cin >> entcard;
          for (j=0; j<=17; j++)
               if (entcard[0]==charrank[j])
                    deal[i].r=intrank[j];
          for (j=0; j<=7; j++)
               if (entcard[1]==charsuit[j])

                    deal[i].s=intsuit[j];

    }
}


void sort(card t[]) // smallest to largest
{
     int i,j,holdr,holds;
     for (i=3; i>=0; i--)
          for (j=0; j<=i; j++)
               if (t[j].r>t[j+1].r)
               {
                    holdr=t[j].r;
                    holds=t[j].s;
                    t[j].r=t[j+1].r;
                    t[j].s=t[j+1].s;
                    t[j+1].r=holdr;
                    t[j+1].s=holds;
               }
 }


 int score(card t[])
 {
     int straight=0;
     int flush=0;
     if (t[0].r==t[3].r)
     {
          if (t[0].r==12)
               return 9;
          else if ((t[0].r>=0)&&(t[0].r<=2))
          {
               if ((t[4].r<=2)||(t[4].r==12))
                    return 8;
               else
                    return 8;

          }
          else
               return 7;

     }
     else if (t[1].r==t[4].r)
     {
          if (t[4].r==12)
          {
```

```
                if (t[0].r<=2)
                        return 9;
                else
                        return 9;
        }
        else if ((t[4].r>=0)&&(t[4].r<=2))
        {
                if ((t[0].r<=2)||(t[0].r==12))
                        return 8;
                else
                        return 8;
        }
        else
                return 7;
    }
    else if ((t[0].r==t[1].r)&&(t[3].r==t[4].r) &&
    ((t[1].r==t[2].r)||(t[2].r==t[3].r)))
        return 6; // full house
    else if ((t[0].r==t[2].r)||(t[1].r==t[3].r)||(t[2].r==t[4].r))
        return 3; // three of a kind
    else if (((t[0].r==t[1].r)&&(t[2].r==t[3].r)) ||
    ((t[0].r==t[1].r)&&(t[3].r==t[4].r)) ||
    ((t[1].r==t[2].r)&&(t[3].r==t[4].r)))
        return 2; // two pair
    else if (((t[0].r==t[1].r)&&(t[0].r>=9))||
        ((t[1].r==t[2].r)&&(t[1].r>=9))||
        ((t[2].r==t[3].r)&&(t[2].r>=9))||
        ((t[3].r==t[4].r)&&(t[3].r>=9)))
        return 1; // pair
    else
    {
        if
((t[0].s==t[1].s)&&(t[1].s==t[2].s)&&(t[2].s==t[3].s)&&(t[3].s==t[4].s)
)
                flush=1;
        if (((t[4].r==(t[0].r+4)) || ((t[3].r==3)&&(t[4].r==12)))
                &&
(t[0].r!=t[1].r)&&(t[2].r!=t[1].r)&&(t[2].r!=t[3].r)&&(t[3].r!=t[4].r))
                straight=1;
        if ((flush==1)&&(straight==1))
        {
                if (t[0].r==8)
                        return 11; // royal flush
                else
                        return 10; // straight flush
        }
        else if (flush==1)
                return 5;
        else if (straight==1)
                return 4;
        else
                return 0;
    }
}
```

```
int samenum(int c1, int c2, int c3, int c4)
{
      if ((c1==c2)&&(c2==c3)&&(c3==c4))
            return 1;
      else
            return 0;
}


int samenum(int c1, int c2, int c3)
{
      if ((c1==c2)&&(c2==c3))
            return 1;
      else
            return 0;
}


void TestHands()
{
      card player[5];
      setrank(0,1,2,3,4,player);
      setsuit(2,1,1,1,1,player,1);
/*    setrank(0,2,4,6,11,player);
      setsuit(1,2,3,4,3,player,1);
      setrank(0,4,4,6,12,player);
      setsuit(1,2,3,4,1,player,1);
      setrank(0,2,4,6,12,player);
      setsuit(1,2,3,4,1,player,1);
      setrank(0,2,4,6,9,player);
      setsuit(1,2,3,4,1,player,1);
      setrank(0,2,4,6,10,player);
      setsuit(1,2,3,4,1,player,1);
      setrank(0,2,4,6,11,player);
      setsuit(1,2,3,4,1,player,1);
      setrank(0,2,4,6,7,player);
      setsuit(1,2,3,4,1,player,1);
      setrank(0,2,4,6,8,player);
      setsuit(1,2,3,4,1,player,1);  */
}


void presetup1()
{
      int i,j,k,l,m;
      card player[5];
      InitPair=false;

      // Set all hands with no matching rank (65637 total)
      for (i=0; i<=8; i++)
      {
            for (j=i+1; j<=9; j++)
            {
                  for (k=j+1; k<=10; k++)
                  {
                        for (l=k+1; l<=11; l++)
                        {
```

```
for (m=l+1; m<=12; m++)
{
      // all suits equal
      setrank(i,j,k,l,m,player);

      setsuit(1,1,1,1,1,player,4);

      // all suits equal except one card
      setsuit(2,1,1,1,1,player,12);
      setsuit(1,2,1,1,1,player,12);
      setsuit(1,1,2,1,1,player,12);
      setsuit(1,1,1,2,1,player,12);
      setsuit(1,1,1,1,2,player,12);

      // three cards one suit, two cards
another
      setsuit(2,2,1,1,1,player,12);

      setsuit(2,1,2,1,1,player,12);

      setsuit(2,1,1,2,1,player,12);

      setsuit(2,1,1,1,2,player,12);

      setsuit(1,2,2,1,1,player,12);

      setsuit(1,2,1,2,1,player,12);

      setsuit(1,2,1,1,2,player,12);

      setsuit(1,1,2,2,1,player,12);

      setsuit(1,1,2,1,2,player,12);

      setsuit(1,1,1,2,2,player,12);

      // three cards one suit, fourth and
fifth two other ranks
      setsuit(2,3,1,1,1,player,24);

      setsuit(2,1,3,1,1,player,24);

      setsuit(2,1,1,3,1,player,24);

      setsuit(2,1,1,1,3,player,24);

      setsuit(1,2,3,1,1,player,24);

      setsuit(1,2,1,3,1,player,24);

      setsuit(1,2,1,1,3,player,24);

      setsuit(1,1,2,3,1,player,24);

      setsuit(1,1,2,1,3,player,24);

      setsuit(1,1,1,2,3,player,24);
```

```
                                        // two cards one suit, two cards
another, fifth card a third
                                        setsuit(1,1,2,2,3,player,24);
                                        setsuit(1,2,1,2,3,player,24);
                                        setsuit(1,2,2,1,3,player,24);
                                        setsuit(1,1,2,3,2,player,24);
                                        setsuit(1,2,1,3,2,player,24);
                                        setsuit(1,2,2,3,1,player,24);
                                        setsuit(1,1,3,2,2,player,24);
                                        setsuit(1,2,3,1,2,player,24);
                                        setsuit(1,2,3,2,1,player,24);
                                        setsuit(1,3,1,2,2,player,24);
                                        setsuit(1,3,2,1,2,player,24);
                                        setsuit(1,3,2,2,1,player,24);
                                        setsuit(3,1,1,2,2,player,24);
                                        setsuit(3,1,2,1,2,player,24);
                                        setsuit(3,1,2,2,1,player,24);
                                        // two cards one suit, all other
suits have one card each
                                        setsuit(4,4,1,2,3,player,24);
                                        setsuit(4,1,4,2,3,player,24);
                                        setsuit(4,2,3,4,1,player,24);
                                        setsuit(4,1,2,3,4,player,24);
                                        setsuit(1,4,4,2,3,player,24);
                                        setsuit(1,4,2,4,3,player,24);
                                        setsuit(1,4,2,3,4,player,24);
                                        setsuit(2,3,4,4,1,player,24);
                                        setsuit(2,3,4,1,4,player,24);
                                        setsuit(1,2,3,4,4,player,24);
                                    }
                                }
                            }
                        }
```

```
	}
}


void presetup2()
{
	int i,j,k,l;
	card player[5];
	InitPair=true;

	// Pairs (91520 total)
	for (i=0; i<=9; i++)
	{
		for (j=i+1; j<=10; j++)
		{
			for (k=j+1; k<=11; k++)
			{
				for (l=k+1; l<=12; l++)
				{
					setrank(i,i,j,k,l,player);

					setsuit(1,2,1,1,1,player,12);
					setsuit(1,2,1,1,2,player,12);
					setsuit(1,2,1,2,1,player,12);
					setsuit(1,2,2,1,1,player,12);
					setsuit(1,2,1,1,3,player,24);
					setsuit(1,2,1,3,1,player,24);
					setsuit(1,2,3,1,1,player,24);
					setsuit(1,2,1,3,3,player,24);
					setsuit(1,2,3,1,3,player,24);
					setsuit(1,2,3,3,1,player,24);
					setsuit(1,2,3,3,3,player,12);
					setsuit(1,2,1,2,3,player,24);
					setsuit(1,2,1,3,2,player,24);
					setsuit(1,2,3,1,2,player,24);
					setsuit(1,2,3,4,4,player,24);
					setsuit(1,2,4,3,4,player,12);
					setsuit(1,2,4,4,3,player,12);
					setsuit(1,2,1,3,4,player,12);
					setsuit(1,2,3,1,4,player,24);
					setsuit(1,2,3,4,1,player,24);

					setrank(i,j,j,k,l,player);

					setsuit(1,1,2,1,1,player,12);
					setsuit(1,1,2,1,2,player,12);
					setsuit(1,1,2,2,1,player,12);
					setsuit(2,1,2,1,1,player,12);
					setsuit(1,1,2,1,3,player,24);
					setsuit(1,1,2,3,1,player,24);
					setsuit(3,1,2,1,1,player,24);
					setsuit(1,1,2,3,3,player,24);
					setsuit(3,1,2,1,3,player,24);
					setsuit(3,1,2,3,1,player,24);
					setsuit(3,1,2,3,3,player,12);
					setsuit(1,1,2,2,3,player,24);
					setsuit(1,1,2,3,2,player,24);
```

```
setsuit(3,1,2,1,2,player,24);
setsuit(3,1,2,4,4,player,24);
setsuit(4,1,2,3,4,player,12);
setsuit(4,1,2,4,3,player,12);
setsuit(1,1,2,3,4,player,12);
setsuit(3,1,2,1,4,player,24);
setsuit(3,1,2,4,1,player,24);


setrank(i,j,k,k,l,player);

setsuit(1,1,1,2,1,player,12);
setsuit(1,1,1,2,2,player,12);

setsuit(1,2,1,2,1,player,12);
setsuit(2,1,1,2,1,player,12);
setsuit(1,1,1,2,3,player,24);
setsuit(1,3,1,2,1,player,24);
setsuit(3,1,1,2,1,player,24);
setsuit(1,3,1,2,3,player,24);
setsuit(3,1,1,2,3,player,24);
setsuit(3,3,1,2,1,player,24);
setsuit(3,3,1,2,3,player,12);
setsuit(1,2,1,2,3,player,24);
setsuit(1,3,1,2,2,player,24);
setsuit(3,1,1,2,2,player,24);
setsuit(3,4,1,2,4,player,24);
setsuit(4,3,1,2,4,player,12);
setsuit(4,4,1,2,3,player,12);
setsuit(1,3,1,2,4,player,12);
setsuit(3,1,1,2,4,player,24);
setsuit(3,4,1,2,1,player,24);


setrank(i,j,k,l,l,player);

setsuit(1,1,1,1,2,player,12);
setsuit(1,1,2,1,2,player,12);
setsuit(1,2,1,1,2,player,12);
setsuit(2,1,1,1,2,player,12);
setsuit(1,1,3,1,2,player,24);
setsuit(1,3,1,1,2,player,24);
setsuit(3,1,1,1,2,player,24);
setsuit(1,3,3,1,2,player,24);
setsuit(3,1,3,1,2,player,24);
setsuit(3,3,1,1,2,player,24);
setsuit(3,3,3,1,2,player,12);
setsuit(1,2,3,1,2,player,24);
setsuit(1,3,2,1,2,player,24);
setsuit(3,1,2,1,2,player,24);
setsuit(3,4,4,1,2,player,24);
setsuit(4,3,4,1,2,player,12);
setsuit(4,4,3,1,2,player,12);
setsuit(1,3,4,1,2,player,12);
setsuit(3,1,4,1,2,player,24);
setsuit(3,4,1,1,2,player,24);
```

```
                        }
                    }
                }
            }
        }

void presetup3()
{
      int i,j,k;
      card player[5];
      InitPair=true;

      // Set all two pair hands (6864 total)
      for (i=0; i<=10; i++)
      {
            for (j=i+1; j<=11; j++)
            {
                  for (k=j+1; k<=12; k++)
                  {
                        setrank(i,i,j,j,k,player);
                        setsuit(1,2,3,4,1,player,12);
                        setsuit(1,2,3,4,3,player,12);
                        setsuit(1,2,1,3,1,player,24);
                        setsuit(1,2,1,3,2,player,24);
                        setsuit(1,2,1,3,3,player,24);
                        setsuit(1,2,1,3,4,player,24);
                        setsuit(1,2,1,2,1,player,12);
                        setsuit(1,2,1,2,3,player,12);

                        setrank(i,i,j,k,k,player);
                        setsuit(1,2,1,3,4,player,12);
                        setsuit(1,2,3,3,4,player,12);
                        setsuit(1,2,1,1,3,player,24);
                        setsuit(1,2,2,1,3,player,24);
                        setsuit(1,2,3,1,3,player,24);
                        setsuit(1,2,4,1,3,player,24);
                        setsuit(1,2,1,1,2,player,12);
                        setsuit(1,2,3,1,2,player,12);

                        setrank(i,j,j,k,k,player);
                        setsuit(1,1,2,3,4,player,12);
                        setsuit(3,1,2,3,4,player,12);
                        setsuit(1,1,2,1,3,player,24);
                        setsuit(2,1,2,1,3,player,24);
                        setsuit(3,1,2,1,3,player,24);
                        setsuit(4,1,2,1,3,player,24);
                        setsuit(1,1,2,1,2,player,12);
                        setsuit(3,1,2,1,2,player,12);
                  }
            }
      }
}
```

```
void presetup4()
{
      int i,j,k;
      card player[5];
      InitPair=true;

      // Set all three of a kind hands (4290 total)
      for (i=0; i<=10; i++)
      {
            for (j=i+1; j<=11; j++)
            {
                  for (k=j+1; k<=12; k++)
                  {
                        setrank(i,i,i,j,k,player);
                        setsuit(1,2,3,1,2,player,24);
                        setsuit(1,2,3,1,4,player,12);
                        setsuit(1,2,3,4,1,player,12);
                        setsuit(1,2,3,1,1,player,12);
                        setsuit(1,2,3,4,4,player,4);

                        setrank(i,j,j,j,k,player);
                        setsuit(1,1,2,3,2,player,24);
                        setsuit(1,1,2,3,4,player,12);
                        setsuit(4,1,2,3,1,player,12);
                        setsuit(1,1,2,3,1,player,12);
                        setsuit(4,1,2,3,4,player,4);

                        setrank(i,j,k,k,k,player);
                        setsuit(1,2,1,2,3,player,24);
                        setsuit(1,4,1,2,3,player,12);
                        setsuit(4,1,1,2,3,player,12);
                        setsuit(1,1,1,2,3,player,12);
                        setsuit(4,4,1,2,3,player,4);

                  }
            }
      }
}


void presetup5()
{
      int i,j;
      card player[5];
      InitPair=true;

      // Set all full house hands (312 total)
      for (i=0; i<=11; i++)
      {
            for (j=i+1; j<=12; j++)
            {
                  setrank(i,i,j,j,j,player);
                  setsuit(1,2,1,2,3,player,12);
                  setsuit(1,4,1,2,3,player,12);

                  setrank(i,i,i,j,j,player);
                  setsuit(1,2,3,1,2,player,12);
```

```
                setsuit(1,2,3,1,4,player,12);
            }
        }
}


void presetup6() // Set all four of a kind hands (156 total)
{
        int i,j;
        card player[5];
        InitPair=true;

        for (i=0; i<=11; i++)
        {
            for (j=i+1; j<=12; j++)
            {

                setrank(i,i,i,i,j,player);
                setsuit(1,2,3,4,1,player,4);

                setrank(i,j,j,j,j,player);
                setsuit(1,1,2,3,4,player,4);
            }
        }
}



void setsuit(int a, int b, int c, int d, int e, card player[], int w)
{
        // Total classes of hands = 134459
        numhand++;
//      if (numhand%36==23)
//      {
            totalw+=w;
            FinHand++;
            player[0].s=a-1;
            player[1].s=b-1;
            player[2].s=c-1;
            player[3].s=d-1;
            player[4].s=e-1;
            playhand1(player,w);
            curtime=time(NULL);
            cerr << numhand << "\t";
            cerr << "hours rem. " << (float)(curtime-
begtime)*(float)(134459*(1.0/36.0)-FinHand)/(float)FinHand/3600 <<
"\n";
//      }
}



void setrank(int a, int b, int c, int d, int e, card player[])
```

```
{
      player[0].r=a;
      player[1].r=b;
      player[2].r=c;
      player[3].r=d;
      player[4].r=e;
}


void playhand1(card deal[], int w)
{
      int
c1,c2,c3,i,r,s,count,KeepNum,DiscNum,maxkeep,total[15],OneRank;
      __int64 TotalInt[15],MaxTotalInt[15];
      double expret,maxret,total2[15],maxtotal2[15];
      card keep[5],disc[5],deck[47],draw1[5];
      count=0;
      for (i=0; i<=14; i++)
            maxtotal2[i]=0;
      for (i=0; i<=4; i++)
            cerr << deal[i].r << "/" << deal[i].s << " ";
      for (r=0; r<=12; r++)
      {
            for (s=0; s<=3; s++)
            {
                  if (((r!=deal[0].r)||(s!=deal[0].s))&&
                        ((r!=deal[1].r)||(s!=deal[1].s))&&
                        ((r!=deal[2].r)||(s!=deal[2].s))&&
                        ((r!=deal[3].r)||(s!=deal[3].s))&&
                        ((r!=deal[4].r)||(s!=deal[4].s)))
                  {
                        deck[count].r=r;
                        deck[count].s=s;
                        count++;
                  }
            }
      }
      if (count!=47)
            cerr << "deck error in playhand1\n";

      maxret=0;
      for (count=0; count<=31; count++)
      {
            KeepNum=0;
            DiscNum=0;
            if (count%32<16)
            {
                  keep[KeepNum].r=deal[0].r;
                  keep[KeepNum].s=deal[0].s;
                  KeepNum++;
            }
            else
            {
                  disc[DiscNum].r=deal[0].r;
                  disc[DiscNum].s=deal[0].s;
                  DiscNum++;
```

```
}
if (count%16<8)
{
      keep[KeepNum].r=deal[1].r;
      keep[KeepNum].s=deal[1].s;
      KeepNum++;
}

else
{
      disc[DiscNum].r=deal[1].r;
      disc[DiscNum].s=deal[1].s;
      DiscNum++;
}
if (count%8<4)
{
      keep[KeepNum].r=deal[2].r;
      keep[KeepNum].s=deal[2].s;
      KeepNum++;
}
else
{
      disc[DiscNum].r=deal[2].r;
      disc[DiscNum].s=deal[2].s;
      DiscNum++;
}
if (count%4<2)
{
      keep[KeepNum].r=deal[3].r;
      keep[KeepNum].s=deal[3].s;
      KeepNum++;
}
else
{
      disc[DiscNum].r=deal[3].r;
      disc[DiscNum].s=deal[3].s;
      DiscNum++;
}
if (count%2<1)
{
      keep[KeepNum].r=deal[4].r;
      keep[KeepNum].s=deal[4].s;
      KeepNum++;
}
else
{
      disc[DiscNum].r=deal[4].r;
      disc[DiscNum].s=deal[4].s;
      DiscNum++;
}
for (i=DiscNum; i<=4; i++)
{
      disc[i].r=99;
      disc[i].s=99;
}
for (i=0; i<=14; i++)
      total[i]=0;
```

```
if (details==1)
{
      for (i=0; i<KeepNum; i++)
            cerr << keep[i].r << "/" << keep[i].s << "\t";
}

expret=0;
for (i=0; i<=14; i++)
{
      total2[i]=0;
      TotalInt[i]=0;
}
if (KeepNum==0)
{
      for (c1=0; c1<=42; c1++)
/*    {
            for (c2=c1+1; c2<=43; c2++)
            {
                  for (c3=c2+1; c3<=44; c3++)
                  {
                        for (c4=c3+1; c4<=45; c4++)
                        {
                              for (c5=c4+1; c5<=46; c5++)
                              {
                                    draw1[0]=deck[c1];
                                    draw1[1]=deck[c2];
                                    draw1[2]=deck[c3];
                                    draw1[3]=deck[c4];
                                    draw1[4]=deck[c5];

expret+=advicesetup(draw1,disc,total);
                              }
                        }
                  }
            }
      }
      expret/=1533939; */
      expret=0.0;
}
else if (KeepNum==1)
{
      int c4;
/*    for (c1=0; c1<=43; c1++)
      {
            for (c2=c1+1; c2<=44; c2++)
            {
                  for (c3=c2+1; c3<=45; c3++)
                  {
                        for (c4=c3+1; c4<=46; c4++)
                        {
                              draw1[0]=deck[c1];
                              draw1[1]=deck[c2];
                              draw1[2]=deck[c3];
                              draw1[3]=deck[c4];
                              draw1[4]=keep[0];

expret+=advicesetup(draw1,disc,total);
```

```
                                        for (i=0; i<=14; i++)
                                        {
                                //              cerr << total[i] << ",

";
      total2[i]+=(double)total[i]/(double)total[14];
                                        }
                                    }
                                }
                            }
                        }
                        expret/=178365;  */
                        expret=Draw4CardsEV[keep[0].r];
                        for (i=0; i<=13; i++)
                              total2[i]=Draw4CardsProb[keep[0].r][i];
 //
 //


                    }
                    else if ((KeepNum==2)&&
                          ((keep[0].r==keep[1].r)||
      ((InitPair==false)&&((keep[0].s==keep[1].s)||abs(keep[0].r-
 keep[1].r)==1))))
                    {
                          for (c1=0; c1<=44; c1++)
                          {
                                for (c2=c1+1; c2<=45; c2++)
                                {
                                      for (c3=c2+1; c3<=46; c3++)
                                      {
                                            draw1[0]=deck[c1];
                                            draw1[1]=deck[c2];
                                            draw1[2]=deck[c3];
                                            draw1[3]=keep[0];
                                            draw1[4]=keep[1];

        expret+=advicesetup(draw1,disc,total);
                                                  for (i=0; i<=14; i++) //
   1086008=combin(44,3) total combinations
                                                  {
   //
        total2[i]+=(double)total[i]/(double)total[14];
                                                        TotalInt[i]+=total[i];
                                                  }
                                            }
                                      }
                                }
                          }
                          expret/=16215;
                          for (i=0; i<=13; i++)
                                total2[i]/=16215;
    //
    //
                    }
                    else if (KeepNum==3)
                    {
                          for (c1=0; c1<=45; c1++)
                          {
                                for (c2=c1+1; c2<=46; c2++)
                                {
```

```
                                    draw1[0]=deck[c1];
                                    draw1[1]=deck[c2];
                                    draw1[2]=keep[0];
                                    draw1[3]=keep[1];
                                    draw1[4]=keep[2];
                                    expret+=advicesetup(draw1,disc,total);
                                    for (i=0; i<=13; i++) //
1086008=combin(45,5)*10 total combinations
                                    {
//
      total2[i]+=(double)total[i]/(double)total[14];
                                          TotalInt[i]+=total[i];
                                    }
                              }
                        }
                        expret/=1081;
//                      for (i=0; i<=13; i++)
//                            total2[i]/=1081;
            }
            else if (KeepNum==4)
            {
                  for (c1=0; c1<=46; c1++)
                  {
                        draw1[0]=deck[c1];
                        draw1[1]=keep[0];
                        draw1[2]=keep[1];
                        draw1[3]=keep[2];
                        draw1[4]=keep[3];
                        expret+=advicesetup(draw1,disc,total);
                        for (i=0; i<=13; i++) //
13707540=combin(46,5)*10 total combinations
                        {
//
      total2[i]+=(double)total[i]/(double)total[14];
                              TotalInt[i]+=total[i];
                        }
                  }
                  expret/=47;
//                for (i=0; i<=13; i++)
//                      total2[i]/=47;
            }
            else if (KeepNum==5)
            {
                  draw1[0]=keep[0];
                  draw1[1]=keep[1];
                  draw1[2]=keep[2];
                  draw1[3]=keep[3];
                  draw1[4]=keep[4];
                  expret=advicesetup(draw1,disc,total);
                  for (i=0; i<=13; i++) // 7667695 = combin(47,5)*5
total combinations
                  {
//                      total2[i]+=(double)total[i]/(double)total[14];
                        TotalInt[i]+=total[i];
                  }
            }
            else
```

```
                expret=0.0;
            if (details==1)
                cerr << expret << "\n";


            if (expret>maxret)
            {
                maxret=expret;
                bestplay=count;
                maxkeep=KeepNum;
                OneRank=keep[0].r;
                for (i=0; i<=13; i++)
                {
//                  maxtotal2[i]=total2[i];
                    MaxTotalInt[i]=TotalInt[i];
                }
            }
        }
//      for (i=0; i<=13; i++)
//          gtotal[i]+=w*maxtotal2[i];
/*      for (i=0; i<=13; i++)
        {
            if (w*MaxTotalInt[i]<0)
            {
                printf("Maxtotal[%i]=%I64i\n",i,MaxTotalInt[i]);
                printf("w*Maxtotal[%i]=%I64i\n",i,MaxTotalInt[i]*w);
                for (r=0; r<=4; r++)
                    cerr << deal[r].r << "/" << deal[r].s << "\t";
                cerr << "\n";
                cin >> s;
            }
        } */


        if (maxkeep==0)
        {
            for (i=0; i<=13; i++)
                Draw5Tot[i]+=w*MaxTotalInt[i];
            Draw5Count+=w;
        }
        else if (maxkeep==1)
        {
            Draw4Tot[OneRank]++;
            Draw4Count+=w;
        }

        else if (maxkeep==2)
        {
            for (i=0; i<=13; i++)
                Draw3Tot[i]+=w*MaxTotalInt[i];
            Draw3Count+=w;
        }
        else if (maxkeep==3)
        {
            for (i=0; i<=13; i++)
                Draw2Tot[i]+=w*MaxTotalInt[i];
            Draw2Count+=w;
```

```
	}
	else if (maxkeep==4)
	{
		for (i=0; i<=13; i++)
			Draw1Tot[i]+=w*MaxTotalInt[i];
		Draw1Count+=w;
	}
	else if (maxkeep==5)
	{
		for (i=0; i<=13; i++)
			Draw0Tot[i]+=w*MaxTotalInt[i];
		Draw0Count+=w;
	}

	TotRet+=w*maxret;
//	cerr << "hand ev = " << maxret << ", gave ev = " <<
TotRet/(double)totalw << "\t";
/*	if (FinHand%1000==1)
	{
		for (i=0; i<=13; i++)
			cout << pay[i] << "\t" << gtotal[i] << "\n";
		cout << "Game ER = \t" << TotRet/(double)totalw << "\n";
	} */
}



/*
void Draw4Cards(void)
{
	int c1,c2,c3,c4,i,j,r,s,rank,count,total[15],MatchRank;
	double expret,gtotal[13][15];
	card disc[5],deck[51],draw1[5];
	double
Deck51Weight[4]={0.977465542,1.072828034,1.175002132,1.284304656};
	unsigned __int64
match0[13][15],match1[13][15],match2[13][15],match3[13][15];


//	0.977465542 = (135751/178365)/(194580/249900) = (combin(47-
3,4)/combin(47,4))/(combin(51-3,4)/combin(51,4))
//	1.072828034 = (39732/178365)/(51888/249900)
//	1.175002132 = (2838/178365)/(3384/249900)
//	1.284304656 = (44/178365)/(48/249900)


	for (i=0; i<=4; i++)
	{
		disc[i].r=99;
		disc[i].s=99;
	}
	for (i=0; i<=12; i++)
		for (j=0; j<=14; j++)
		{
			gtotal[i][j]=0;
			match0[i][j]=0;
			match1[i][j]=0;
```

```
                match2[i][j]=0;
                match3[i][j]=0;
        }
for (rank=0; rank<=12; rank++)
{
        expret=0;
        cout << "rank = \t" << rank << "\t";
        cerr << "rank = \t" << rank << "\t";
        count=0;
        for (r=0; r<=12; r++) // deck setup
        {
                for (s=0; s<=3; s++)
                {
                        if ((r!=rank)||(s!=0))
                        {
                                deck[count].r=r;
                                deck[count].s=s;
                                count++;
                        }
                }
        }
        if (count!=51)
                cerr << "Error 200\n";
        for (c1=0; c1<=47; c1++)
        {
                for (c2=c1+1; c2<=48; c2++)
                {
                        for (c3=c2+1; c3<=49; c3++)
                        {
                                for (c4=c3+1; c4<=50; c4++)
                                {
                                        for (i=0; i<=14; i++)
                                                total[i]=0;

    MatchRank=(deck[c1].r==rank?1:0)+(deck[c2].r==rank?1:0)+(deck[c3]
.r==rank?1:0)+(deck[c4].r==rank?1:0);
                                        draw1[0]=deck[c1];
                                        draw1[1]=deck[c2];
                                        draw1[2]=deck[c3];
                                        draw1[3]=deck[c4];
                                        draw1[4].r=rank;
                                        draw1[4].s=0;
                                        sort(draw1);

    expret+=advicesetup(draw1,disc,total)*Deck51Weight[MatchRank];
                                        for (i=0; i<=13; i++)
                                        {

    gtotal[rank][i]+=((float)total[i]/(float)total[14])*Deck51Weight[
MatchRank];                                     if (MatchRank==0)

    match0[rank][i]+=total[i];                  else if (MatchRank==1)

    match1[rank][i]+=total[i];                  else if (MatchRank==2)
```

```
            match2[rank][i]+=total[i];
                                                else if (MatchRank==3)

            match3[rank][i]+=total[i];
                                          }
                                    }
                              }
                        }
                  }
                  cout << expret << "\n";
                  cerr << expret << "\n";
            }
            for (rank=0; rank<=12; rank++)
            {
                  printf("Rank=\t%i\t",rank);
                  for (i=0; i<=13; i++)
                        printf("%f\t",gtotal[rank][i]);
                  printf("\n");
            }
            printf("Match0\n");
            for (rank=0; rank<=12; rank++)
            {
                  printf("Rank=\t%i\t",rank);
                  for (i=0; i<=13; i++)
                        printf("%I64i\t",match0[rank][i]);
                  printf("\n");
            }
            printf("Match1\n");
            for (rank=0; rank<=12; rank++)
            {
                  printf("Rank=\t%i\t",rank);
                  for (i=0; i<=13; i++)
                        printf("%I64i\t",match1[rank][i]);
                  printf("\n");
            }
            printf("Match2\n");
            for (rank=0; rank<=12; rank++)
            {
                  printf("Rank=\t%i\t",rank);
                  for (i=0; i<=13; i++)
                        printf("%I64i\t",match2[rank][i]);
                  printf("\n");
            }
            printf("Match3\n");
            for (rank=0; rank<=12; rank++)
            {
                  printf("Rank=\t%i\t",rank);
                  for (i=0; i<=13; i++)
                        printf("%I64i\t",match3[rank][i]);
                  printf("\n");
            }
      }   */


      void Draw4Cards(void)
```

```
{
      int
r1,r2,r3,r4,p1,p2,p3,p4,i,j,r,s,rank,count,total[15],DiscCom,NumInitHan
ds,TotCom;
      double expret,gtotal[13][15];
      card disc[5],deck1[51],deck2[47],draw1[5];
      unsigned __int64 gtotal_int[13][15];


      for (i=0; i<=12; i++)
            for (j=0; j<=14; j++)
            {
                  gtotal[i][j]=0;
                  gtotal_int[i][j]=0;
            }
      for (rank=0; rank<=12; rank++)
      {
            TotCom=0;
            NumInitHands=0;
            expret=0;
            cout << "rank = \t" << rank << "\t";
            cerr << "rank = \t" << rank << "\t";
            count=0;
            for (r=0; r<=12; r++) // deck setup
            {
                  for (s=0; s<=3; s++)
                  {
                        if ((r!=rank)||(s!=0))
                        {
                              deck1[count].r=r;
                              deck1[count].s=s;
                              count++;
                        }
                  }
            }
            if (count!=51)
                  cerr << "Error 200\n";
            DiscCom=0;
            for (r1=0; r1<=9; r1++)
            {
                  for (r2=r1+1; r2<=10; r2++)
                  {
                        for (r3=r2+1; r3<=11; r3++)
                        {
                              for (r4=r3+1; r4<=12; r4++)
                              {
                                    if
((rank!=r1)&&(rank!=r2)&&(rank!=r3)&&(rank!=r4))
                                    {
                                          DiscCom++;
                                          if (DiscCom%3==2)
                                          {
                                                NumInitHands++;
                                                cerr << rank << "\t" <<
r1 << ", " << r2 << ", " << r3 << ", " << r4 << "\n";
                                                count=0;
```

```
// deck setup

s++)

(((r!=rank)||(s!=0))&&
      ((r!=r1)||(s!=1))&&
      ((r!=r2)||(s!=2))&&
      ((r!=r3)||(s!=3))&&
      ((r!=r4)||(s!=1)))

      deck2[count].r=r;
      deck2[count].s=s;
      count++;



201\n";



p1++)

p2<=44; p2++)

(p3=p2+1; p3<=45; p3++)

(p4=p3+1; p4<=46; p4++)

      TotCom++;
      for (i=0; i<=14; i++)
      total[i]=0;

for (r=0; r<=12; r++)

{
      for (s=0; s<=3;

      {
            if

                  {

                  }
      }
}
if (count!=47)
      cerr << "Error

disc[0].r=r1;
disc[1].r=r2;
disc[2].r=r3;
disc[3].r=r4;
disc[0].s=1;
disc[1].s=2;
disc[2].s=3;
disc[3].s=1;

for (p1=0; p1<=43;

{
      for (p2=p1+1;

      {
            for

            {
                  for

                  {
```

```
draw1[0]=deck2[p1];

draw1[1]=deck2[p2];

draw1[2]=deck2[p3];

draw1[3]=deck2[p4];

draw1[4].r=rank;

draw1[4].s=0;

sort(draw1);

expret+=advicesetup(draw1,disc,total);
                                                                    //

cerr << total[14] << "\n";

for (i=0; i<=13; i++)

{

gtotal[rank][i]+=((float)total[i]/(float)total[14]);

gtotal_int[rank][i]+=total[i];

}
                                                                        }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
      }
      printf("Rank=\t%i\n",rank);
      printf("Number Initial Hands=\t%i\n",NumInitHands);
      printf("Total Combinations=\t%i\n",TotCom);
      printf("Exptected Return=\t%f\n",expret);
      cerr << expret << "\t" << expret/(float)TotCom << "\n";
}
printf("Pay Table\n");
for (i=0; i<=13; i++)
      printf("%s\t%i\n",pokerhands[i],i);

for (rank=0; rank<=12; rank++)
{
      printf("Rank=\t%i\t",rank);
      for (i=0; i<=13; i++)
            printf("%f\t",gtotal[rank][i]);
      printf("\n");
}
printf("Total Combinations\n");
for (rank=0; rank<=12; rank++)
```

```
	{
		printf("Rank=\t%i\t",rank);
		for (i=0; i<=13; i++)
			printf("%I64i\t",gtotal_int[rank][i]);
		printf("\n");
	}
}



void fast0(int total[])
{
	int i,j,k,l,numcom;

	for (i=0; i<=7; i++) // straight flush
		for (int j=0; j<=3; j++)

	total[10]+=indeck[i][j]*indeck[i+1][j]*indeck[i+2][j]*indeck[i+3]
[j]*indeck[i+4][j];

	for (j=0; j<=3; j++)
	{

	total[10]+=indeck[12][j]*indeck[0][j]*indeck[1][j]*indeck[2][j]*i
ndeck[3][j];

	total[11]+=indeck[8][j]*indeck[9][j]*indeck[10][j]*indeck[11][j]*
indeck[12][j];
	}

	for (i=0; i<=8; i++)

	total[4]+=indeck[i][4]*indeck[i+1][4]*indeck[i+2][4]*indeck[i+3][
4]*indeck[i+4][4];
	total[4]+=indeck[12][4]*indeck[0][4]*indeck[1][4]*indeck[2][4]*in
deck[3][4];

	for (i=0; i<=3; i++)
	{
		if (indeck[13][i]==13)
			total[5]+=1287;
		else if (indeck[13][i]==12)
			total[5]+=792;
		else if (indeck[13][i]==11)
			total[5]+=462;
		else if (indeck[13][i]==10)
			total[5]+=252;
		else if (indeck[13][i]==9)
			total[5]+=126;
		else if (indeck[13][i]==8)
			total[5]+=56;
		else if (indeck[13][i]==7)
			total[5]+=21;
		else if (indeck[13][i]==6)
			total[5]+=6;
		else if (indeck[13][i]==5)
			total[5]+=1;
```

```
	}
	total[4]-=(total[10]+total[11]);
	total[5]-=(total[10]+total[11]);


	if (indeck[12][4]==4) // four aces
	{
		total[9]+=indeck[0][4]+indeck[1][4]+indeck[2][4];

	total[9]+=indeck[3][4]+indeck[4][4]+indeck[5][4]+indeck[6][4]+ind
eck[7][4]+indeck[8][4]+indeck[9][4]+indeck[10][4]+indeck[11][4];
	}
	if (indeck[0][4]==4) // four 2's
	{
		total[8]+=indeck[12][4]+indeck[1][4]+indeck[2][4];

	total[8]+=indeck[3][4]+indeck[4][4]+indeck[5][4]+indeck[6][4]+ind
eck[7][4]+indeck[8][4]+indeck[9][4]+indeck[10][4]+indeck[11][4];
	}
	if (indeck[1][4]==4) // four 3's
	{
		total[8]+=indeck[12][4]+indeck[0][4]+indeck[2][4];

	total[8]+=indeck[3][4]+indeck[4][4]+indeck[5][4]+indeck[6][4]+ind
eck[7][4]+indeck[8][4]+indeck[9][4]+indeck[10][4]+indeck[11][4];
	}
	if (indeck[2][4]==4) // four 4's
	{
		total[8]+=indeck[12][4]+indeck[0][4]+indeck[1][4];

	total[8]+=indeck[3][4]+indeck[4][4]+indeck[5][4]+indeck[6][4]+ind
eck[7][4]+indeck[8][4]+indeck[9][4]+indeck[10][4]+indeck[11][4];
	}
	for (i=3; i<=11; i++)
		if (indeck[i][4]==4)
			total[7]+=43;

	/* full house */
	for (i=0; i<=11; i++)
	{
		for (j=i+1; j<=12; j++)
		{
			if ((indeck[i][4]==2)&&(indeck[j][4]==3))
				total[6]++;
			else if ((indeck[i][4]==3)&&(indeck[j][4]==2))
				total[6]+=1;
			else if ((indeck[i][4]==2)&&(indeck[j][4]==4))
				total[6]+=4;
			else if ((indeck[i][4]==4)&&(indeck[j][4]==2))
				total[6]+=4;
			else if ((indeck[i][4]==3)&&(indeck[j][4]==3))
				total[6]+=6;
			else if ((indeck[i][4]==3)&&(indeck[j][4]==4))
				total[6]+=18; /* 4c2+3*4 */
			else if ((indeck[i][4]==4)&&(indeck[j][4]==3))
				total[6]+=18;
			else if ((indeck[i][4]==4)&&(indeck[j][4]==4))
```

```
                        total[6]+=48; /* 2*(4*4c2) */
            }
}


/* three of a kind */
for (i=0; i<=12; i++)
{
      if (indeck[i][4]==4)
            total[3]+=3612; /* 4*43c2 */
      else if (indeck[i][4]==3)
            total[3]+=946; /* 44c2 */
}

total[3]-=total[6];

/* two pair */
for (i=0; i<=11; i++)
{
      for (j=i+1; j<=12; j++)
      {
            if ((indeck[i][4]==2)&&(indeck[j][4]==2))
                  total[2]+=43;

            else if ((indeck[i][4]==2)&&(indeck[j][4]==3))
                  total[2]+=126; /* 3*42 */
            else if ((indeck[i][4]==3)&&(indeck[j][4]==2))
                  total[2]+=126; /* 3*42 */

            else if ((indeck[i][4]==2)&&(indeck[j][4]==4))
                  total[2]+=246; /* 6*41 */
            else if ((indeck[i][4]==4)&&(indeck[j][4]==2))
                  total[2]+=246; /* 6*41 */

            else if ((indeck[i][4]==3)&&(indeck[j][4]==3))
                  total[2]+=369; /* 3*3*41 */

            else if ((indeck[i][4]==3)&&(indeck[j][4]==4))
                  total[2]+=720; /* 3*6*40 */
            else if ((indeck[i][4]==4)&&(indeck[j][4]==3))
                  total[2]+=720; /* 3*6*40 */

            else if ((indeck[i][4]==4)&&(indeck[j][4]==4))
                  total[2]+=1404; /* 6*6*39 */
      }
}


/* pair */
for (i=9; i<=12; i++)
{
      if (indeck[i][4]==4)
            numcom=6;
      else if (indeck[i][4]==3)
            numcom=3;
      else if (indeck[i][4]==2)
```

```
                numcom=1;

            if (indeck[i][4]>=2)
                for (j=0; j<=10; j++)
                    for (k=j+1; k<=11; k++)
                        for (l=k+1; l<=12; l++)
                            if ((i!=j)&&(i!=k)&&(i!=l))

    total[1]+=(numcom*indeck[j][4]*indeck[k][4]*indeck[l][4]);
    }


    total[0]=Draw2Combin[0];
    for (i=1; i<=13; i++)
        total[0]-=total[i];
    if (total[5]<0)
        cerr << "total[5]=\t" << total[5] << "\n";
}




void fast1(int r, int s, int total[])
{
    int i,j,k,l,numcom;

    if (r>=4)
        if (r!=12)
            total[10]+=indeck[r-4][s]*indeck[r-3][s]*indeck[r-
2][s]*indeck[r-1][s];
        else
            total[11]=indeck[r-4][s]*indeck[r-3][s]*indeck[r-
2][s]*indeck[r-1][s];
    if ((r>=3)&&(r<=11))
        if (r!=11)
            total[10]+=indeck[r-3][s]*indeck[r-2][s]*indeck[r-
1][s]*indeck[r+1][s];
        else
            total[11]=indeck[r-3][s]*indeck[r-2][s]*indeck[r-
1][s]*indeck[r+1][s];
    if ((r>=2)&&(r<=10))
        if (r!=10)
            total[10]+=indeck[r-2][s]*indeck[r-
1][s]*indeck[r+1][s]*indeck[r+2][s];
        else
            total[11]=indeck[r-2][s]*indeck[r-
1][s]*indeck[r+1][s]*indeck[r+2][s];
    if ((r>=1)&&(r<=9))
        if (r!=9)
            total[10]+=indeck[r-
1][s]*indeck[r+1][s]*indeck[r+2][s]*indeck[r+3][s];
        else
            total[11]=indeck[r-
1][s]*indeck[r+1][s]*indeck[r+2][s]*indeck[r+3][s];
    if (r<=8)
        if (r!=8)
```

```
      total[10]+=indeck[r+1][s]*indeck[r+2][s]*indeck[r+3][s]*indeck[r+
4][s];
            else

      total[11]=indeck[r+1][s]*indeck[r+2][s]*indeck[r+3][s]*indeck[r+4
][s];

      if (r==12)

      total[10]+=indeck[0][s]*indeck[1][s]*indeck[2][s]*indeck[3][s];
      else if (r==0)

      total[10]+=indeck[12][s]*indeck[1][s]*indeck[2][s]*indeck[3][s];
      else if (r==1)

      total[10]+=indeck[12][s]*indeck[0][s]*indeck[2][s]*indeck[3][s];
      else if (r==2)

      total[10]+=indeck[12][s]*indeck[0][s]*indeck[1][s]*indeck[3][s];
      else if (r==3)

      total[10]+=indeck[12][s]*indeck[0][s]*indeck[1][s]*indeck[2][s];


      if (r>=4)
            total[4]+=indeck[r-4][4]*indeck[r-3][4]*indeck[r-
2][4]*indeck[r-1][4];
      if ((r>=3)&&(r<=11))
            total[4]+=indeck[r-3][4]*indeck[r-2][4]*indeck[r-
1][4]*indeck[r+1][4];
      if ((r>=2)&&(r<=10))
            total[4]+=indeck[r-2][4]*indeck[r-
1][4]*indeck[r+1][4]*indeck[r+2][4];
      if ((r>=1)&&(r<=9))
            total[4]+=indeck[r-
1][4]*indeck[r+1][4]*indeck[r+2][4]*indeck[r+3][4];
      if (r<=9)

      total[4]+=indeck[r+1][4]*indeck[r+2][4]*indeck[r+3][4]*indeck[r+4
][4];

      if (r==12)

      total[4]+=indeck[0][4]*indeck[1][4]*indeck[2][4]*indeck[3][4];
      else if (r==0)

      total[4]+=indeck[12][4]*indeck[1][4]*indeck[2][4]*indeck[3][4];
      else if (r==1)

      total[4]+=indeck[12][4]*indeck[0][4]*indeck[2][4]*indeck[3][4];
      else if (r==2)

      total[4]+=indeck[12][4]*indeck[0][4]*indeck[1][4]*indeck[3][4];
      else if (r==3)

      total[4]+=indeck[12][4]*indeck[0][4]*indeck[1][4]*indeck[2][4];
```

```
total[4]-=(total[10]+total[11]);

if (indeck[13][s]==12)
      total[5]=495;
else if (indeck[13][s]==11)
      total[5]=330;
else if (indeck[13][s]==10)
      total[5]=210;
else if (indeck[13][s]==9)
      total[5]=126;
else if (indeck[13][s]==8)
      total[5]=70;
else if (indeck[13][s]==7)
      total[5]=35;
else if (indeck[13][s]==6)
      total[5]=15;
else if (indeck[13][s]==5)
      total[5]=5;
else if (indeck[13][s]==4)
      total[5]=1;

total[5]-=(total[10]+total[11]);



for (i=0; i<=12; i++)
{
      if ((i!=r)&&(indeck[i][4]==4)) // four draw cards are equal
      {
            if (i==12)
            {
                  if (r<=2)
                        total[9]++;
                  else
                        total[9]++;
            }
            else if (i<=2)
            {
                  if ((r<=2)||(r==12))
                        total[8]++;
                  else
                        total[8]++;
            }
            else
                  total[7]++;
      }
      if ((i!=r)&&(indeck[r][4]==3))
      {
            if ((r==12)&&(i<=2)) // four aces + 2-4
                  total[9]+=indeck[i][4];
            else if (r==12) // four aces + 5-K
                  total[9]+=indeck[i][4];
            else if ((r<=2)&&((i<=2)||(i==12))) // four 2-4 + A-4
                  total[8]+=indeck[i][4];
            else if (r<=2) // four 2-4 + 5-K
```

```
                        total[8]+=indeck[i][4];
                  else // four 5-K
                        total[7]+=indeck[i][4];
            }
      }

      /* full house */
      for (i=0; i<=12; i++)
      {
            if (i!=r)
            {
                  if (indeck[r][4]==3)
                  {
                        if (indeck[i][4]==4)
                              total[6]+=30;
                        else if (indeck[i][4]==3)
                              total[6]+=12;
                        else if (indeck[i][4]==2)
                              total[6]+=3;
                  }
                  else if (indeck[r][4]==2)
                  {
                        if (indeck[i][4]==4)
                              total[6]+=14;
                        else if (indeck[i][4]==3)
                              total[6]+=5;
                        else if (indeck[i][4]==2)
                              total[6]++;
                  }
                  else if (indeck[r][4]==1)
                  {
                        if (indeck[i][4]==4)
                              total[6]+=4;
                        else if (indeck[i][4]==3)
                              total[6]++;
                  }
            }
      }

      /* three of a kind */
      for (i=0; i<=11; i++)
      {
            for (j=i+1; j<=12; j++)
            {
                  if ((i!=r)&&(j!=r))
                  {
                        if (indeck[i][4]==4)
                              total[3]+=4*indeck[j][4];
                        else if (indeck[i][4]==3)
                              total[3]+=indeck[j][4];

                        if (indeck[j][4]==4)
                              total[3]+=4*indeck[i][4];
                        else if (indeck[j][4]==3)
                              total[3]+=indeck[i][4];

                        if (indeck[r][4]==3)
```

```
                        total[3]+=(3*indeck[i][4]*indeck[j][4]);
                  else if (indeck[r][4]==2)
                        total[3]+=(indeck[i][4]*indeck[j][4]);
            }
      }
}


/* two pair */
for (i=0; i<=11; i++)
{
      for (j=i+1; j<=12; j++)
      {
            if ((i!=r)&&(j!=r))
            {
                  if ((indeck[i][4]==2)&&(indeck[j][4]==2))
                  {
                        total[2]++;
                        total[2]+=(indeck[r][4]*4);
                  }

                  else if ((indeck[i][4]==2)&&(indeck[j][4]==3))
                  {
                        total[2]+=3;
                        total[2]+=(indeck[r][4]*9);
                  }
                  else if ((indeck[i][4]==3)&&(indeck[j][4]==2))
                  {
                        total[2]+=3;
                        total[2]+=(indeck[r][4]*9);
                  }
                  else if ((indeck[i][4]==2)&&(indeck[j][4]==4))
                  {
                        total[2]+=6;
                        total[2]+=(indeck[r][4]*16);
                  }
                  else if ((indeck[i][4]==4)&&(indeck[j][4]==2))
                  {
                        total[2]+=6;
                        total[2]+=(indeck[r][4]*16);
                  }
                  else if ((indeck[i][4]==3)&&(indeck[j][4]==3))
                  {
                        total[2]+=9;
                        total[2]+=(indeck[r][4]*18);
                  }
                  else if ((indeck[i][4]==3)&&(indeck[j][4]==4))
                  {
                        total[2]+=18;
                        total[2]+=(indeck[r][4]*30);
                  }
                  else if ((indeck[i][4]==4)&&(indeck[j][4]==3))
                  {
                        total[2]+=18;
                        total[2]+=(indeck[r][4]*30);
                  }
```

```
                        else if ((indeck[i][4]==4)&&(indeck[j][4]==4))
                        {
                              total[2]+=36;
                              total[2]+=(indeck[r][4]*48);
                        }
                        else if ((indeck[i][4]==1)&&(indeck[j][4]==4))
                              total[2]+=(indeck[r][4]*6);
                        else if ((indeck[i][4]==1)&&(indeck[j][4]==3))
                              total[2]+=(indeck[r][4]*3);
                        else if ((indeck[i][4]==1)&&(indeck[j][4]==2))
                              total[2]+=(indeck[r][4]);
                        else if ((indeck[i][4]==4)&&(indeck[j][4]==1))
                              total[2]+=(indeck[r][4]*6);
                        else if ((indeck[i][4]==3)&&(indeck[j][4]==1))
                              total[2]+=(indeck[r][4]*3);
                        else if ((indeck[i][4]==2)&&(indeck[j][4]==1))
                              total[2]+=(indeck[r][4]);
                  }
            }
      }


/* pair */
for (j=0; j<=10; j++)
      for (k=j+1; k<=11; k++)
            for (l=k+1; l<=12; l++)
                  if ((r!=j)&&(r!=k)&&(r!=l))
                  {
                        if (r>=9)

total[1]+=(indeck[r][4]*indeck[j][4]*indeck[k][4]*indeck[l][4]);

                        if (j>=9)
                        {
                              if (indeck[j][4]==4)
                                    numcom=6;
                              else if (indeck[j][4]==3)
                                    numcom=3;
                              else if (indeck[j][4]==2)
                                    numcom=1;
                              else
                                    numcom=0;

total[1]+=(numcom*indeck[k][4]*indeck[l][4]);
                        }

                        if (k>=9)
                        {
                              if (indeck[k][4]==4)
                                    numcom=6;
                              else if (indeck[k][4]==3)
                                    numcom=3;
                              else if (indeck[k][4]==2)
                                    numcom=1;
                              else
                                    numcom=0;
```

```
        total[1]+=(numcom*indeck[j][4]*indeck[l][4]);
                                }

                                if (l>=9)
                                {
                                        if (indeck[l][4]==4)
                                                numcom=6;
                                        else if (indeck[l][4]==3)
                                                numcom=3;
                                        else if (indeck[l][4]==2)
                                                numcom=1;
                                        else
                                                numcom=0;

        total[1]+=(numcom*indeck[j][4]*indeck[k][4]);
                                }
                        }

        total[0]=Draw2Combin[1];
        for (i=1; i<=13; i++)
                total[0]-=total[i];
}

void fast2(card t[], int total[])
{
        int i,j,k,hold;
        int r1=t[0].r;
        int r2=t[1].r;
        int s1=t[0].s;
        int s2=t[1].s;
        if (r2<r1)
        {
                hold=r1;
                r1=r2;
                r2=hold;
                hold=s1;
                s1=s2;
                s2=hold;
        }

        /* straight and royal flush */
        if ((r1+1==r2)&&(s1==s2))
        {
                if (r1>=3)
                        if (r2!=12)
                                total[10]+=indeck[r1-3][s1]*indeck[r1-
2][s1]*indeck[r1-1][s1];
                        else
                                total[11]=indeck[r1-3][s1]*indeck[r1-
2][s1]*indeck[r1-1][s1];
                if ((r1>=2)&&(r2<=11))
                        if (r2!=11)
                                total[10]+=indeck[r1-2][s1]*indeck[r1-
1][s1]*indeck[r2+1][s1];
                        else
```

```
                        total[11]=indeck[r1-2][s1]*indeck[r1-
1][s1]*indeck[r2+1][s1];
            if ((r1>=1)&&(r2<=10))
                  if (r2!=10)
                        total[10]+=indeck[r1-
1][s1]*indeck[r2+1][s1]*indeck[r2+2][s1];
                  else
                        total[11]=indeck[r1-
1][s1]*indeck[r2+1][s1]*indeck[r2+2][s1];
            if (r2<=9)
                  if (r2!=9)

      total[10]+=indeck[r2+1][s1]*indeck[r2+2][s1]*indeck[r2+3][s1];
                  else

      total[11]=indeck[r2+1][s1]*indeck[r2+2][s1]*indeck[r2+3][s1];
      }
      else if ((r1+2==r2)&&(s1==s2))
      {
            if (r1>=2)
                  if (r2!=12)
                        total[10]+=indeck[r1-2][s1]*indeck[r1-
1][s1]*indeck[r1+1][s1];
                  else // (10)-(j)-q-(k)-a

      total[11]=indeck[8][s1]*indeck[9][s1]*indeck[11][s1];
            if ((r1>=1)&&(r2<=11))
                  if (r2!=11)
                        total[10]+=indeck[r1-
1][s1]*indeck[r1+1][s1]*indeck[r2+1][s1];
                  else // (10)-j-(q)-k-(a)

      total[11]=indeck[8][s1]*indeck[10][s1]*indeck[12][s1];
            if (r2<=10)
                  if (r2!=10)

      total[10]+=indeck[r1+1][s1]*indeck[r2+1][s1]*indeck[r2+2][s1];
                  else // 10-(j)-q-(k)-(a)

      total[11]=indeck[9][s1]*indeck[11][s1]*indeck[12][s1];
      }
      else if ((r1+3==r2)&&(s1==s2))
      {
            if (r1>=1)
                  if (r2!=12)
                        total[10]+=indeck[r1-
1][s1]*indeck[r1+1][s1]*indeck[r1+2][s1];
                  else
                        total[11]=indeck[r1-
1][s1]*indeck[r1+1][s1]*indeck[r1+2][s1];
            if (r2<=11)
                  if (r2!=11)

      total[10]+=indeck[r1+1][s1]*indeck[r1+2][s1]*indeck[r2+1][s1];
                  else

      total[11]=indeck[r1+1][s1]*indeck[r1+2][s1]*indeck[r2+1][s1];
```

```
        }
        else if ((r1+4==r2)&&(s1==s2))
        {
              if (r2!=12)

        total[10]+=indeck[r1+1][s1]*indeck[r1+2][s1]*indeck[r1+3][s1];
              else

        total[11]=indeck[r1+1][s1]*indeck[r1+2][s1]*indeck[r1+3][s1];
        }

        if ((r1==0)&&(r2==12)&&(s1==s2))
              total[10]+=indeck[1][s1]*indeck[2][s1]*indeck[3][s1];
        else if ((r1==1)&&(r2==12)&&(s1==s2))
              total[10]+=indeck[0][s1]*indeck[2][s1]*indeck[3][s1];
        else if ((r1==2)&&(r2==12)&&(s1==s2))
              total[10]+=indeck[0][s1]*indeck[1][s1]*indeck[3][s1];
        else if ((r1==3)&&(r2==12)&&(s1==s2))
              total[10]+=indeck[0][s1]*indeck[1][s1]*indeck[2][s1];
        else if ((r1==0)&&(r2==1)&&(s1==s2))
              total[10]+=indeck[2][s1]*indeck[3][s1]*indeck[12][s1];
        else if ((r1==0)&&(r2==2)&&(s1==s2))
              total[10]+=indeck[1][s1]*indeck[3][s1]*indeck[12][s1];
        else if ((r1==0)&&(r2==3)&&(s1==s2))
              total[10]+=indeck[1][s1]*indeck[2][s1]*indeck[12][s1];
        else if ((r1==1)&&(r2==2)&&(s1==s2))
              total[10]+=indeck[0][s1]*indeck[3][s1]*indeck[12][s1];
        else if ((r1==1)&&(r2==3)&&(s1==s2))
              total[10]+=indeck[0][s1]*indeck[2][s1]*indeck[12][s1];
        else if ((r1==2)&&(r2==3)&&(s1==s2))
              total[10]+=indeck[0][s1]*indeck[1][s1]*indeck[12][s1];


        /* straight */

        if (r1+1==r2)
        {
              if (r1>=3)
                    total[4]+=indeck[r1-3][4]*indeck[r1-2][4]*indeck[r1-
1][4];
              if ((r1>=2)&&(r2<=11))
                    total[4]+=indeck[r1-2][4]*indeck[r1-
1][4]*indeck[r2+1][4];
              if ((r1>=1)&&(r2<=10))
                    total[4]+=indeck[r1-
1][4]*indeck[r2+1][4]*indeck[r2+2][4];
              if (r2<=9)

        total[4]+=indeck[r2+1][4]*indeck[r2+2][4]*indeck[r2+3][4];
        }
        else if (r1+2==r2)
        {
              if (r1>=2)
                    total[4]+=indeck[r1-2][4]*indeck[r1-
1][4]*indeck[r1+1][4];
              if ((r1>=1)&&(r2<=11))
```

```
                  total[4]+=indeck[r1-
1][4]*indeck[r1+1][4]*indeck[r2+1][4];
            if (r2<=10)

      total[4]+=indeck[r1+1][4]*indeck[r2+1][4]*indeck[r2+2][4];
      }
      else if (r1+3==r2)
      {
            if (r1>=1)
                  total[4]+=indeck[r1-
1][4]*indeck[r1+1][4]*indeck[r1+2][4];
            if (r2<=11)

      total[4]+=indeck[r1+1][4]*indeck[r1+2][4]*indeck[r2+1][4];
      }
      else if (r1+4==r2)
      {
            total[4]+=indeck[r1+1][4]*indeck[r1+2][4]*indeck[r1+3][4];
      }

      if ((r1==0)&&(r2==12))
            total[4]+=indeck[1][4]*indeck[2][4]*indeck[3][4];
      else if ((r1==1)&&(r2==12))
            total[4]+=indeck[0][4]*indeck[2][4]*indeck[3][4];
      else if ((r1==2)&&(r2==12))
            total[4]+=indeck[0][4]*indeck[1][4]*indeck[3][4];
      else if ((r1==3)&&(r2==12))
            total[4]+=indeck[0][4]*indeck[1][4]*indeck[2][4];
      else if ((r1==0)&&(r2==1))
            total[4]+=indeck[2][4]*indeck[3][4]*indeck[12][4];
      else if ((r1==0)&&(r2==2))
            total[4]+=indeck[1][4]*indeck[3][4]*indeck[12][4];
      else if ((r1==0)&&(r2==3))
            total[4]+=indeck[1][4]*indeck[2][4]*indeck[12][4];
      else if ((r1==1)&&(r2==2))
            total[4]+=indeck[0][4]*indeck[3][4]*indeck[12][4];
      else if ((r1==1)&&(r2==3))
            total[4]+=indeck[0][4]*indeck[2][4]*indeck[12][4];
      else if ((r1==2)&&(r2==3))
            total[4]+=indeck[0][4]*indeck[1][4]*indeck[12][4];


      total[4]-=(total[10]+total[11]);

      if (s1==s2)
      {
            if (indeck[13][s1]==11)
                  total[5]=165;
            else if (indeck[13][s1]==10)
                  total[5]=120;
            else if (indeck[13][s1]==9)
                  total[5]=84;
            else if (indeck[13][s1]==8)
                  total[5]=56;
            else if (indeck[13][s1]==7)
                  total[5]=35;
            else if (indeck[13][s1]==6)
```

```
            total[5]=20;
      else if (indeck[13][s1]==5)
            total[5]=10;
      else if (indeck[13][s1]==4)
            total[5]=4;
      else if (indeck[13][s1]==3)
            total[5]=1;
}

total[5]-=(total[10]+total[11]);

/* four of a kind */
for (i=0; i<=12; i++)
{
      if ((r1==r2)&&(i!=r1)&&(indeck[r1][4]==2))
      {
            if (r1==12)
            {
                  if (i<=2)
                        total[9]+=indeck[i][4];
                  else
                        total[9]+=indeck[i][4];
            }
            else if (r1<=2)
            {
                  if ((i<=2)||(i==12))
                        total[8]+=indeck[i][4];
                  else
                        total[8]+=indeck[i][4];
            }
            else
                  total[7]+=indeck[i][4];
      }
      else if ((r1!=r2)&&(i==r1)&&(indeck[r1][4]==3))
      {
            if (r1==12)
            {
                  if (r2<=2)
                        total[9]++;
                  else
                        total[9]++;
            }
            else if (r1<=2)
            {
                  if ((r2<=2)||(r2==12))
                        total[8]++;
                  else
                        total[8]++;
            }
            else
                  total[7]++;
      }
      else if ((r1!=r2)&&(i==r2)&&(indeck[r2][4]==3))
      {
            if (r2==12)
```

```
            {
                  if (r1<=2)
                        total[9]++;
                  else
                        total[9]++;
            }
            else if (r2<=2)
            {
                  if ((r1<=2)||(r1==12))
                        total[8]++;
                  else
                        total[8]++;
            }
            else
                  total[7]++;
      }
}

/* full house */
if (r1==r2)
{
      for (i=0; i<=12; i++)
      {
            if (i!=r1)
            {
                  if (indeck[i][4]==4)
                  {
                        total[6]+=4;
                        if (indeck[r1][4]==2)
                              total[6]+=12;
                        else if (indeck[r1][4]==1)
                              total[6]+=6;
                  }
                  else if (indeck[i][4]==3)
                  {
                        total[6]++;
                        if (indeck[r1][4]==2)
                              total[6]+=6;
                        else if (indeck[r1][4]==1)
                              total[6]+=3;
                  }
                  else if (indeck[i][4]==2)

                  {
                        if (indeck[r1][4]==2)
                              total[6]+=2;
                        else if (indeck[r1][4]==1)
                              total[6]++;
                  }
            }
      }
}
else
{
      if ((indeck[r1][4]==3)&&(indeck[r2][4]==3))
            total[6]=18;
```

```
      else if ((indeck[r1][4]==3)&&(indeck[r2][4]==2))

            total[6]=9;
      else if ((indeck[r1][4]==2)&&(indeck[r2][4]==3))
            total[6]=9;
      else if ((indeck[r1][4]==3)&&(indeck[r2][4]==1))
            total[6]=3;
      else if ((indeck[r1][4]==1)&&(indeck[r2][4]==3))
            total[6]=3;
      else if ((indeck[r1][4]==2)&&(indeck[r2][4]==2))
            total[6]=4;
      else if ((indeck[r1][4]==2)&&(indeck[r2][4]==1))
            total[6]=1;
      else if ((indeck[r1][4]==1)&&(indeck[r2][4]==2))
            total[6]=1;
}



/* three of a kind */
if (r1!=r2)
{
      for (i=0; i<=12; i++)
      {
            if ((i!=r1)&&(i!=r2))
            {
                  if (indeck[i][4]==4)
                        total[3]+=4;
                  else if (indeck[i][4]==3)
                        total[3]++;

                  if (indeck[r1][4]==3)
                        total[3]+=3*indeck[i][4];
                  else if (indeck[r1][4]==2)
                        total[3]+=indeck[i][4];

                  if (indeck[r2][4]==3)
                        total[3]+=3*indeck[i][4];
                  else if (indeck[r2][4]==2)
                        total[3]+=indeck[i][4];
            }
      }
}
else
      for (i=0; i<=11; i++)
            for (j=i+1; j<=12; j++)
                  if ((i!=r1)&&(j!=r1))

total[3]+=indeck[r1][4]*indeck[i][4]*indeck[j][4];



/* two pair */
if (r1!=r2)
{
      for (i=0; i<=12; i++)
```

```
        {
                if ((i!=r1)&&(i!=r2))
                {

total[2]+=indeck[i][4]*indeck[r1][4]*indeck[r2][4];

                        if (indeck[i][4]==4)

total[2]+=6*(indeck[r1][4]+indeck[r2][4]);

                        else if (indeck[i][4]==3)

total[2]+=3*(indeck[r1][4]+indeck[r2][4]);
                        if (indeck[i][4]==2)
                                total[2]+=(indeck[r1][4]+indeck[r2][4]);
                }
        }
}
else
        for (i=0; i<=11; i++)
                for (j=i+1; j<=12; j++)
                        if ((i!=r1)&&(j!=r1))
                        {
                                if (indeck[i][4]==4)
                                        total[2]+=(6*indeck[j][4]);
                                else if (indeck[i][4]==3)
                                        total[2]+=(3*indeck[j][4]);
                                else if (indeck[i][4]==2)
                                        total[2]+=indeck[j][4];

                                if (indeck[j][4]==4)
                                        total[2]+=(6*indeck[i][4]);
                                else if (indeck[j][4]==3)
                                        total[2]+=(3*indeck[i][4]);
                                else if (indeck[j][4]==2)
                                        total[2]+=indeck[i][4];
                        }


/* high pair */
if ((r1!=r2)&&(r1>=9)&&(r2>=9))
{
        for (i=0; i<=11; i++)
        {
                for (j=i+1; j<=12; j++)
                {
                        if ((i!=r1)&&(i!=r2)&&(j!=r1)&&(j!=r2))
                        {

total[1]+=((indeck[r1][4]+indeck[r2][4])*indeck[i][4]*indeck[j][4
]);

                                if (i>=9)
                                        if (indeck[i][4]==4)
                                                total[1]+=(6*indeck[j][4]);
                                        else if (indeck[i][4]==3)
```

```
                                    total[1]+=(3*indeck[j][4]);
                              else if (indeck[i][4]==2)
                                    total[1]+=indeck[j][4];

                        if (j>=9)
                              if (indeck[j][4]==4)
                                    total[1]+=(6*indeck[i][4]);
                              else if (indeck[j][4]==3)
                                    total[1]+=(3*indeck[i][4]);
                              else if (indeck[j][4]==2)
                                    total[1]+=indeck[i][4];
                  }
            }
      }
}

else if ((r1!=r2)&&(r1>=9)&&(r2<=8))
{
      for (i=0; i<=11; i++)
      {
            for (j=i+1; j<=12; j++)
            {
                  if ((i!=r1)&&(i!=r2)&&(j!=r1)&&(j!=r2))
                  {

total[1]+=(indeck[r1][4]*indeck[i][4]*indeck[j][4]);

                        if (i>=9)
                              if (indeck[i][4]==4)
                                    total[1]+=(6*indeck[j][4]);
                              else if (indeck[i][4]==3)
                                    total[1]+=(3*indeck[j][4]);
                              else if (indeck[i][4]==2)
                                    total[1]+=indeck[j][4];

                        if (j>=9)
                              if (indeck[j][4]==4)
                                    total[1]+=(6*indeck[i][4]);
                              else if (indeck[j][4]==3)
                                    total[1]+=(3*indeck[i][4]);
                              else if (indeck[j][4]==2)
                                    total[1]+=indeck[i][4];
                  }
            }

      }
}


else if ((r1!=r2)&&(r1<=8)&&(r2>=9))
{
      for (i=0; i<=11; i++)
      {
            for (j=i+1; j<=12; j++)
            {
                  if ((i!=r1)&&(i!=r2)&&(j!=r1)&&(j!=r2))
                  {
```

```
total[1]+=(indeck[r2][4]*indeck[i][4]*indeck[j][4]);

                        if (i>=9)
                        {
                              if (indeck[i][4]==4)
                                    total[1]+=(6*indeck[j][4]);
                              else if (indeck[i][4]==3)
                                    total[1]+=(3*indeck[j][4]);
                              else if (indeck[i][4]==2)
                                    total[1]+=indeck[j][4];
                        }
                        if (j>=9)
                        {
                              if (indeck[j][4]==4)
                                    total[1]+=(6*indeck[i][4]);
                              else if (indeck[j][4]==3)
                                    total[1]+=(3*indeck[i][4]);
                              else if (indeck[j][4]==2)
                                    total[1]+=indeck[i][4];
                        }
                  }
            }
      }
}


else if ((r1!=r2)&&(r1<=8)&&(r2<=8))
{
      for (i=0; i<=11; i++)
      {
            for (j=i+1; j<=12; j++)
            {
                  if ((i!=r1)&&(i!=r2)&&(j!=r1)&&(j!=r2))
                  {
                        if (i>=9)
                        {
                              if (indeck[i][4]==4)
                                    total[1]+=(6*indeck[j][4]);
                              else if (indeck[i][4]==3)
                                    total[1]+=(3*indeck[j][4]);
                              else if (indeck[i][4]==2)
                                    total[1]+=indeck[j][4];
                        }
                        if (j>=9)
                        {
                              if (indeck[j][4]==4)
                                    total[1]+=(6*indeck[i][4]);
                              else if (indeck[j][4]==3)
                                    total[1]+=(3*indeck[i][4]);
                              else if (indeck[j][4]==2)
                                    total[1]+=indeck[i][4];
                        }
                  }
            }
      }
}
```

```
        else if ((r1==r2)&&(r1>=9))
        {
                for (i=0; i<=10; i++)
                {
                        for (j=i+1; j<=11; j++)
                        {
                                for (k=j+1; k<=12; k++)
                                {
                                        if ((i!=r1)&&(j!=r1)&&(k!=r1))

        total[1]+=(indeck[i][4]*indeck[j][4]*indeck[k][4]);
                                }
                        }
                }
        }

        total[0]=Draw2Combin[2];
        for (i=1; i<=13; i++)
                total[0]-=total[i];
}


void fast3(card t[], int total[])
{
        int i,j,hold,flush;

        for (i=1; i>=0; i--)
                for (j=0; j<=i; j++)
                        if (t[j].r > t[j+1].r)
                        {
                                hold=t[j].r;
                                t[j].r=t[j+1].r;
                                t[j+1].r=hold;
                                hold=t[j].s;
                                t[j].s=t[j+1].s;
                                t[j+1].s=hold;
                        }

        int r1=t[0].r;
        int r2=t[1].r;
        int r3=t[2].r;
        int s1=t[0].s;
        int s2=t[1].s;
        int s3=t[2].s;

        if ((s1==s2)&&(s2==s3))
                flush=1;
        else
                flush=0;



        if (flush==1)
        {
                /* royal flush */
```

```
if ((r1==8)&&(r2==9)&&(r3==10))
      total[11]=indeck[11][s1]*indeck[12][s1];
else if ((r1==8)&&(r2==9)&&(r3==11))
      total[11]=indeck[10][s1]*indeck[12][s1];
else if ((r1==8)&&(r2==9)&&(r3==12))
      total[11]=indeck[10][s1]*indeck[11][s1];
else if ((r1==8)&&(r2==10)&&(r3==11))
      total[11]=indeck[9][s1]*indeck[12][s1];
else if ((r1==8)&&(r2==10)&&(r3==12))
      total[11]=indeck[9][s1]*indeck[11][s1];
else if ((r1==8)&&(r2==11)&&(r3==12))
      total[11]=indeck[9][s1]*indeck[10][s1];
else if ((r1==9)&&(r2==10)&&(r3==11))
      total[11]=indeck[8][s1]*indeck[12][s1];
else if ((r1==9)&&(r2==10)&&(r3==12))
      total[11]=indeck[8][s1]*indeck[11][s1];
else if ((r1==9)&&(r2==11)&&(r3==12))
      total[11]=indeck[8][s1]*indeck[10][s1];
else if ((r1==10)&&(r2==11)&&(r3==12))
      total[11]=indeck[8][s1]*indeck[9][s1];

/* straight flush */
if ((r2==r1+1)&&(r3==r1+2))
{
      if (r1>=2)
            total[10]=indeck[r1-2][s1]*indeck[r1-1][s1];
      if ((r1>=1)&&(r3<=11))
            total[10]+=indeck[r1-1][s1]*indeck[r3+1][s1];
      if (r3<=10)
            total[10]+=indeck[r3+1][s1]*indeck[r3+2][s1];
}
else if ((r2==r1+2)&&(r3==r1+3))
{
      if (r1>=1)
            total[10]=indeck[r1-1][s1]*indeck[r1+1][s1];
      if (r3<=11)
            total[10]+=indeck[r1+1][s1]*indeck[r3+1][s1];
}
else if ((r2==r1+1)&&(r3==r1+3))
{
      if (r1>=1)
            total[10]=indeck[r1-1][s1]*indeck[r1+2][s1];
      if (r3<=11)
            total[10]+=indeck[r1+2][s1]*indeck[r3+1][s1];
}
else if ((r2==r1+1)&&(r3==r1+4))
      total[10]=indeck[r1+2][s1]*indeck[r1+3][s1];
else if ((r2==r1+2)&&(r3==r1+4))
      total[10]=indeck[r1+1][s1]*indeck[r1+3][s1];
else if ((r2==r1+3)&&(r3==r1+4))
      total[10]=indeck[r1+1][s1]*indeck[r1+2][s1];


if ((r1==0)&&(r2==1)&&(r3==2))
      total[10]+=indeck[3][s1]*indeck[12][s1];
else if ((r1==0)&&(r2==1)&&(r3==3))
      total[10]+=indeck[2][s1]*indeck[12][s1];
```

```
      else if ((r1==0)&&(r2==2)&&(r3==3))
            total[10]+=indeck[1][s1]*indeck[12][s1];
      else if ((r1==1)&&(r2==2)&&(r3==3))
            total[10]+=indeck[0][s1]*indeck[12][s1];
      else if ((r1==0)&&(r2==1)&&(r3==12))
            total[10]+=indeck[2][s1]*indeck[3][s1];
      else if ((r1==0)&&(r2==2)&&(r3==12))
            total[10]+=indeck[1][s1]*indeck[3][s1];
      else if ((r1==0)&&(r2==3)&&(r3==12))
            total[10]+=indeck[1][s1]*indeck[2][s1];
      else if ((r1==1)&&(r2==2)&&(r3==12))
            total[10]+=indeck[0][s1]*indeck[3][s1];
      else if ((r1==1)&&(r2==3)&&(r3==12))
            total[10]+=indeck[0][s1]*indeck[2][s1];
      else if ((r1==2)&&(r2==3)&&(r3==12))
            total[10]+=indeck[0][s1]*indeck[1][s1];

      total[10]-=total[11];
}


/* straight */

if ((r2==r1+1)&&(r3==r1+2))
{
      if (r1>=2)
            total[4]=indeck[r1-2][4]*indeck[r1-1][4];
      if ((r1>=1)&&(r3<=11))
            total[4]+=(indeck[r1-1][4]*indeck[r3+1][4]);
      if (r3<=10)
            total[4]+=(indeck[r3+1][4]*indeck[r3+2][4]);
}
else if ((r2==r1+2)&&(r3==r1+3))
{
      if (r1>=1)
            total[4]=indeck[r1-1][4]*indeck[r1+1][4];
      if (r3<=11)
            total[4]+=indeck[r1+1][4]*indeck[r3+1][4];
}
else if ((r2==r1+1)&&(r3==r1+3))
{
      if (r1>=1)
            total[4]=indeck[r1-1][4]*indeck[r1+2][4];
      if (r3<=11)
            total[4]+=indeck[r1+2][4]*indeck[r3+1][4];
}
else if ((r2==r1+1)&&(r3==r1+4))
      total[4]=indeck[r1+2][4]*indeck[r1+3][4];
else if ((r2==r1+2)&&(r3==r1+4))
      total[4]=indeck[r1+1][4]*indeck[r1+3][4];
else if ((r2==r1+3)&&(r3==r1+4))
      total[4]=indeck[r1+1][4]*indeck[r1+2][4];

if ((r1==0)&&(r2==1)&&(r3==2))
      total[4]+=indeck[3][4]*indeck[12][4];
else if ((r1==0)&&(r2==1)&&(r3==3))
      total[4]+=indeck[2][4]*indeck[12][4];
```

```
else if ((r1==0)&&(r2==2)&&(r3==3))
      total[4]+=indeck[1][4]*indeck[12][4];
else if ((r1==1)&&(r2==2)&&(r3==3))
      total[4]+=indeck[0][4]*indeck[12][4];
else if ((r1==0)&&(r2==1)&&(r3==12))
      total[4]+=indeck[2][4]*indeck[3][4];
else if ((r1==0)&&(r2==2)&&(r3==12))
      total[4]+=indeck[1][4]*indeck[3][4];
else if ((r1==0)&&(r2==3)&&(r3==12))
      total[4]+=indeck[1][4]*indeck[2][4];
else if ((r1==1)&&(r2==2)&&(r3==12))
      total[4]+=indeck[0][4]*indeck[3][4];
else if ((r1==1)&&(r2==3)&&(r3==12))
      total[4]+=indeck[0][4]*indeck[2][4];
else if ((r1==2)&&(r2==3)&&(r3==12))
      total[4]+=indeck[0][4]*indeck[1][4];

total[4]-=(total[10]+total[11]);

if (flush==1)
{
      if (indeck[13][s1]==10)
            total[5]=45;
      else if (indeck[13][s1]==9)
            total[5]=36;
      else if (indeck[13][s1]==8)
            total[5]=28;
      else if (indeck[13][s1]==7)
            total[5]=21;
      else if (indeck[13][s1]==6)
            total[5]=15;
      else if (indeck[13][s1]==5)
            total[5]=10;
      else if (indeck[13][s1]==4)
            total[5]=6;
      else if (indeck[13][s1]==3)
            total[5]=3;
      else if (indeck[13][s1]==2)
            total[5]=1;
}

total[5]-=(total[10]+total[11]);


// four of a kind
if ((r1==r3)&&(indeck[r1][4]==1))
{
      if (r1==12)
      {
            for (i=0; i<=11; i++)
                  total[9]+=indeck[i][4];
      }
      else if (r1<=2)
      {
            for (i=0; i<=12; i++)
                  if (i!=r1)
                        total[8]+=indeck[i][4];
```

```
        }
        else
        {
                for (i=0; i<=12; i++)
                        if (i!=r1)
                                total[7]+=indeck[i][4];
        }
}
else if ((r1==r2)&&(indeck[r1][4]==2))
{
        if (r1==12)
        {
                total[9]=1;
        }
        else if (r1<=2)
        {
                total[8]=1;
        }
        else
                total[7]=1;
}
else if ((r2==r3)&&(indeck[r2][4]==2))
{
        if (r2==12)
                total[9]=1;
        else if (r2<=2)
                total[8]=1;
        else
                total[7]=1;
}

/* full house */
if (r1==r3)
{
        for (i=0; i<=12; i++)
        {
                if (i!=r1)
                {
                        if (indeck[i][4]==4)
                                total[6]+=6;
                        else if (indeck[i][4]==3)
                                total[6]+=3;
                        else if (indeck[i][4]==2)
                                total[6]++;
                }
        }
}
else if (r1==r2)
{
        if (indeck[r3][4]==3)
                total[6]+=3;
        else if (indeck[r3][4]==2)
                total[6]++;
        total[6]+=(indeck[r1][4]*indeck[r3][4]);
}
else if (r2==r3)
{
```

```
      if (indeck[r1][4]==3)
            total[6]+=3;
      else if (indeck[r1][4]==2)
            total[6]++;
      total[6]+=(indeck[r1][4]*indeck[r3][4]);
}



/* three of a kind */
if (r1==r3)
{

      for (i=0; i<=11; i++)
            for (j=i+1; j<=12; j++)
                  if ((i!=r1)&&(j!=r1))
                        total[3]+=(indeck[i][4]*indeck[j][4]);
}
else if (r1==r2)
{
      for (i=0; i<=12; i++)
            if ((i!=r1)&&(i!=r3))
                  total[3]+=(indeck[r1][4]*indeck[i][4]);
}
else if (r2==r3)
{
      for (i=0; i<=12; i++)
            if ((i!=r1)&&(i!=r2))
                  total[3]+=(indeck[r2][4]*indeck[i][4]);
}
else
{
      if (indeck[r1][4]==3)
            total[3]+=3;
      else if (indeck[r1][4]==2)
            total[3]++;
      if (indeck[r2][4]==3)
            total[3]+=3;
      else if (indeck[r2][4]==2)
            total[3]++;
      if (indeck[r3][4]==3)
            total[3]+=3;
      else if (indeck[r3][4]==2)
            total[3]++;
}

/* two pair */
if ((r1==r2)&&(r2!=r3))
{
      for (i=0; i<=12; i++)
      {
            if ((i!=r1)&&(i!=r3))
            {
                  total[2]+=(indeck[r3][4]*indeck[i][4]);
                  if (indeck[i][4]==4)
                        total[2]+=6;
                  else if (indeck[i][4]==3)
```

```
                        total[2]+=3;
                    else if (indeck[i][4]==2)
                        total[2]++;
                }
            }
        }
        else if ((r1!=r2)&&(r2==r3))
        {
            for (i=0; i<=12; i++)
            {
                if ((i!=r1)&&(i!=r3))
                {
                    total[2]+=(indeck[r1][4]*indeck[i][4]);
                    if (indeck[i][4]==4)
                        total[2]+=6;
                    else if (indeck[i][4]==3)
                        total[2]+=3;
                    else if (indeck[i][4]==2)
                        total[2]++;
                }
            }
        }
        else if ((r1!=r2)&&(r2!=r3))
        {
            total[2]+=(indeck[r1][4]*indeck[r2][4]);
            total[2]+=(indeck[r1][4]*indeck[r3][4]);
            total[2]+=(indeck[r2][4]*indeck[r3][4]);
        }



        /* high pair */
        if ((r3<=8)&&(r2>r1)&&(r3>r2)) /* three different low cards */
        {
            for (i=9; i<=12; i++)
            {
                if (indeck[i][4]==4)
                    total[1]+=6;
                else if (indeck[i][4]==3)
                    total[1]+=3;
                else if (indeck[i][4]==2)
                    total[1]++;
            }
        }
        else if ((r2>=9)&&(r2>r1)&&(r2==r3)) /* high pair, one lower
    singleton */
        {
            for (i=0; i<=11; i++)
            {
                for (j=i+1; j<=12; j++)
                {
                    if ((i!=r1)&&(i!=r2)&&(j!=r1)&&(j!=r2))
                    {
                        total[1]+=(indeck[i][4]*indeck[j][4]);
                    }
                }
            }
```

```
        }
        else if ((r2>=9)&&(r2<r3)&&(r2==r1)) /* high pair, one higher
singleton */
        {
                for (i=0; i<=11; i++)
                {
                        for (j=i+1; j<=12; j++)
                        {
                                if ((i!=r3)&&(i!=r2)&&(j!=r3)&&(j!=r2))
                                {
                                        total[1]+=(indeck[i][4]*indeck[j][4]);
                                }
                        }
                }
        }
        else if ((r2<=8)&&(r3>=9)&&(r2>r1)) /* one high card, two low */
        {
                for (i=0; i<=12; i++)
                {
                        if ((i!=r1)&&(i!=r2)&&(i!=r3))
                        {
                                total[1]+=(indeck[i][4]*indeck[r3][4]);
                        }
                }
                for (i=9; i<=12; i++)
                {
                        if (i!=r3)
                        {
                                if (indeck[i][4]==4)
                                        total[1]+=6;
                                else if (indeck[i][4]==3)
                                        total[1]+=3;
                                else if (indeck[i][4]==2)
                                        total[1]++;
                        }
                }
        }
        else if ((r1<=8)&&(r2>=9)&&(r3>r2)) /* two high cards, one low */
        {
                for (i=0; i<=12; i++)
                {
                        if ((i!=r1)&&(i!=r2)&&(i!=r3))
                        {
                                total[1]+=(indeck[i][4]*indeck[r2][4]);
                                total[1]+=(indeck[i][4]*indeck[r3][4]);
                        }
                }
                for (i=9; i<=12; i++)
                {
                        if ((i!=r2)&&(i!=r3))
                        {
                                if (indeck[i][4]==4)
                                        total[1]+=6;
                                else if (indeck[i][4]==3)
                                        total[1]+=3;
                                else if (indeck[i][4]==2)
                                        total[1]++;
```

```
                }
            }
        }
        else if ((r1>=9)&&(r2>r1)&&(r3>r2)) /* three high cards */
        {
            for (i=0; i<=12; i++)
            {
                if ((i!=r1)&&(i!=r2)&&(i!=r3))
                {
                    total[1]+=(indeck[i][4]*indeck[r1][4]);
                    total[1]+=(indeck[i][4]*indeck[r2][4]);
                    total[1]+=(indeck[i][4]*indeck[r3][4]);
                }
            }
            for (i=9; i<=12; i++)
            {
                if ((i!=r1)&&(i!=r2)&&(i!=r3))
                {
                    if (indeck[i][4]==4)
                        total[1]+=6;
                    else if (indeck[i][4]==3)
                        total[1]+=3;
                    else if (indeck[i][4]==2)
                        total[1]++;
                }
            }
        }

        total[0]=Draw2Combin[3];
        for (i=1; i<=13; i++)
            total[0]-=total[i];
}



void slow4(card deal[], card deck[], int RemCards, int total[])
{
        card t[5];
        for (int i=0; i<RemCards; i++)
        {
            t[0].r=deal[0].r;
            t[0].s=deal[0].s;
            t[1].r=deal[1].r;
            t[1].s=deal[1].s;
            t[2].r=deal[2].r;
            t[2].s=deal[2].s;
            t[3].r=deal[3].r;
            t[3].s=deal[3].s;
            t[4].r=deck[i].r;
            t[4].s=deck[i].s;
            sort(t);
            total[score(t)]++;
        }
}

void slow5(card t[], int total[])
{
```

```
        sort(t);
        total[score(t)]++;
}

/*

double advicesubmit(int keep, card inhand[], card deck[], int total[])
{
        int i;
        card hand[5];
        for (i=0; i<=10; i++)
                total[i]=0;

        for (i=0; i<=4; i++)
        {
                hand[i].r=inhand[i].r;
                hand[i].s=inhand[i].s;
        }

        if (keep==0)
                fast0(total);
        else if (keep==1)
                fast1(hand[0].r,hand[0].s,total);
        else if (keep==2)
                fast2(hand,total);
        else if (keep==3)
                fast3(hand,total);
        else if (keep==4)
                slow4(hand,deck,total);
        else if (keep==5)
                slow5(hand,total);


        int payoff=0;
        int totret=0;
        for (i=0; i<=9; i++)
                total[10]+=total[i];
        for (i=0; i<=9; i++)
        {
                payoff=total[i]*pay[i];
                totret+=payoff;
        }
        return (double)totret/(double)total[10];
} */


double advicesetup(card deal[], card disc[], int total[])
{
        card hand[5],deck[47];
        double expret;
        double maxret=0;

        int match;
        int tot=0;
        int i,j,k,RemCards;
        int maxkeep=0;
```

```
for (i=0; i<=13; i++)
      for (j=0; j<=4; j++)
            indeck[i][j]=0;

for (i=0; i<=12; i++)
{
      for (j=0; j<=3; j++)
      {
            match=0;
            for (k=0; k<=4; k++)
            {
                  if ((deal[k].r==i) && (deal[k].s==j))
                        match=1;
                  if ((disc[k].r==i) && (disc[k].s==j))
                        match=1;
            }
            if (match==0)
            {
                  deck[tot].r=i;
                  deck[tot].s=j;
                  indeck[i][j]=1;
                  indeck[i][4]++;
                  indeck[13][j]++;
                  tot++;
            }
      }
}
RemCards=tot;
if (RemCards==42)
{
      Draw2Combin[0]=850668;
      Draw2Combin[1]=111930;
      Draw2Combin[2]=11480;
      Draw2Combin[3]=861;
      Draw2Combin[4]=42;
      Draw2Combin[5]=1;
}
else if (RemCards==43)
{
      Draw2Combin[0]=962598;
      Draw2Combin[1]=123410;
      Draw2Combin[2]=12341;
      Draw2Combin[3]=903;
      Draw2Combin[4]=43;
      Draw2Combin[5]=1;
}
else if (RemCards==44)
{
      Draw2Combin[0]=1086008;
      Draw2Combin[1]=135751;
      Draw2Combin[2]=13244;
      Draw2Combin[3]=946;
      Draw2Combin[4]=44;
      Draw2Combin[5]=1;
}
else if (RemCards==45)
{
```

```
        Draw2Combin[0]=1221759;
        Draw2Combin[1]=148995;
        Draw2Combin[2]=14190;
        Draw2Combin[3]=990;
        Draw2Combin[4]=45;
        Draw2Combin[5]=1;
    }
    else if (RemCards==46)
    {
        Draw2Combin[0]=1370754;
        Draw2Combin[1]=163185;
        Draw2Combin[2]=15180;
        Draw2Combin[3]=1035;
        Draw2Combin[4]=46;
        Draw2Combin[5]=1;
    }

    else if (RemCards==47)
    {
        Draw2Combin[0]=1533939;
        Draw2Combin[1]=178365;
        Draw2Combin[2]=16215;
        Draw2Combin[3]=1081;
        Draw2Combin[4]=47;
        Draw2Combin[5]=1;
    }

    bestplay=0;
    int keep,DiscNum,btotal[15];

    for (int count=0; count<=31; count++)
    {
        keep=0;
        DiscNum=4;
        if (count%32<16)
        {
            hand[keep]=deal[0];
            keep++;
        }
        else
        {
            hand[DiscNum]=deal[0];
            DiscNum--;
        }
        if (count%16<8)
        {
            hand[keep]=deal[1];
            keep++;
        }
        else
        {
            hand[DiscNum]=deal[1];
            DiscNum--;
        }
        if (count%8<4)
        {
            hand[keep]=deal[2];
```

```
                        keep++;
                }
                else
                {
                        hand[DiscNum]=deal[2];
                        DiscNum--;
                }
                if (count%4<2)
                {
                        hand[keep]=deal[3];
                        keep++;
                }
                else
                {
                        hand[DiscNum]=deal[3];
                        DiscNum--;
                }
                if (count%2<1)
                {
                        hand[keep]=deal[4];
                        keep++;
                }
                else
                {
                        hand[DiscNum]=deal[4];
                        DiscNum--;
                }
                for (i=0; i<=14; i++)
                        total[i]=0;
                expret=advicesubmit2(keep,hand,deck,count,total,RemCards);
                for (i=0; i<=13; i++)
                        if (total[i]<0)
                        {
                                cerr << "1.  total\t" << i << "\t" << total[i]
<< "keep=\t" << keep <<  "\n";
                                cin >> i;
                        }

                if (expret>maxret)
                {
                        maxret=expret;
                        bestplay=count;
                        maxkeep=keep;
                        for (i=0; i<=14; i++)
                                btotal[i]=total[i];
                }
        }
        for (i=0; i<=14; i++)
        {
                if (RemCards==42)
                        total[i]=weight42[maxkeep]*btotal[i];
                else if (RemCards==43)
                        total[i]=weight43[maxkeep]*btotal[i];
                else if (RemCards==44)
                        total[i]=weight44[maxkeep]*btotal[i];
                else if (RemCards==45)
                        total[i]=weight45[maxkeep]*btotal[i];
```

```
            else if (RemCards==46)
                  total[i]=weight46[maxkeep]*btotal[i];
            else if (RemCards==47)
                  total[i]=weight47[maxkeep]*btotal[i];
            else
                  cerr << "Error 100\n";
      }
/*    cerr << maxkeep << "\t" << btotal[14] << "\t" <<
weight43[maxkeep] << "\n";
      if (btotal[14]*weight43[maxkeep]!=4812990)
            cin >> i; */
      return maxret;
}

double advicesubmit2(int keep, card inhand[], card deck[], int count,
int total[], int RemCards)
{
      int i;
      card hand[5];

      for (i=0; i<keep; i++)
            hand[i]=inhand[i];

      if (keep==0)
            fast0(total);
      else if (keep==1)
            fast1(hand[0].r,hand[0].s,total);
      else if (keep==2)
            fast2(hand,total);
      else if (keep==3)
            fast3(hand,total);
      else if (keep==4)
            slow4(hand,deck,RemCards,total);
      else if (keep==5)
            slow5(hand,total);

      int payoff=0;
      int totret=0;
      for (i=0; i<=13; i++)
      {
            total[14]+=total[i];
            payoff=total[i]*pay[i];
            totret+=payoff;
      }
      return (double)totret/(double)total[14];
}



int strflushrank(int r1, int r2, int r3)
{
      int gaps;
      int hc;
      if ((r1+2==r3)&&(r1>=1)&&(r3<=10))
            gaps=0;
      else if ((r1+3==r3)&&(r3!=12))
```

```
            gaps=1;
      else if ((r3==2)||(r1==9)) // 2-3-4 or J-Q-K
            gaps=1;
      else if (r1+4==r3)
            gaps=2;
      else if ((r3==12)&&((r1>=8)||(r2<=3))) // A high or A low 3 s.f.
            gaps=2;
      else
            return -99;
      if (r1>=9)
            hc=3;
      else if (r2>=9)
            hc=2;
      else if (r3>=9)
            hc=1;
      else
            hc=0;
//    cerr << r1 << ", " << r2 << ", " << r3 << "\t";
//    cerr << "hc = " << hc << "\tgaps = " << gaps << "\n";
      return hc-gaps;
}

int abs(int x)
{
      if (x>=0)
            return x;
      else
            return -1*x;
}

int __fastcall RandNum()
{
      static HCRYPTPROV Provider = NULL;
      int RetValue;


      if (!Provider)
      {
            if (!CryptAcquireContext(&Provider, NULL, NULL,
PROV_RSA_FULL, 0))
            if (!CryptAcquireContext(&Provider, NULL, NULL,
PROV_RSA_FULL,
            CRYPT_NEWKEYSET))
                  return(0);

            RandNum();    // Throw out first number.  Possibly non-
random.
      }

      RetValue = 0;
      if (!CryptGenRandom(Provider, sizeof(int), (unsigned char *)
&RetValue))
            RetValue = 0;
      return(RetValue);
}
```

APPENDIX B

```
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <stdio.h>

struct card
{
  int r;
  int s;
};


void GameReturnSlow(card deck[]);
void GameReturnFast(void);
void TestHand(card deck[]);
float draw(int deal[], int total[], int *TotDiscard);
float PlayHand(card deal[], int weight);
void SetArray(void);
void sort(card d[]);
int score(card t[]);
void entercards(card deal[],int numcards);
float advicesubmit2(int keep, card deal[], card deck[], int count, int
total[]);
void presetup1();
void presetup2();
void presetup3();
void presetup4();
void presetup5();
void presetup6();
void setsuit(int a, int b, int c, int d, int e, card player[], int w);
void setrank(int a, int b, int c, int d, int e, card player[]);

int ScoreArray[2598960],TotalWeight,numhand,begtime,curtime,details;
__int64 TotEvent[20];
float GameEV;
int
DiscardNum[32]={5,4,4,3,4,3,3,2,4,3,3,2,3,2,2,1,4,3,3,2,3,2,2,1,3,2,2,1
,2,1,1,0};
int DrawWeight[6]={7669695,163185,7095,473,43,5};
int pay[14]={0,1,2,3,4,6,9,25,25,25,25,25,50,800}; // jacks or better
// EV=0.995439 SD=4.4175

char
charrank[13]={'2','3','4','5','6','7','8','9','T','J','Q','K','A'};
char charsuit[4]={'h','d','c','s'};



int main()
{
```

```
        int ch;
        cerr << "Start SetArray.\n";
        SetArray();
        cerr << "Finished SetArray.\n";
        do
        {
                cerr << "1.  Game return - Fast\n";
                cerr << "9.  Quit\n";
                cin >> ch;
                if (ch==1)
                        GameReturnFast();
        }
        while (ch!=9);
        return 1;
}



void GameReturnFast(void)
{
        int i;
        details=0;
        TotalWeight=0;
        GameEV=0;
        for (i=0; i<=19; i++)
                TotEvent[i]=0;
        begtime=time(NULL);
        presetup1();
        presetup2();
        presetup3();
        presetup4();
        presetup5();
        presetup6();
        cout << GameEV << "\t" << GameEV/2598960 << "\n";
        cout << "Total hands=\t" << numhand << "\n";
        cout << "Total weight =\t" << TotalWeight << "\n";
        for (i=0; i<=13; i++)
                printf("%i\t%i\t%I64i\n",i,pay[i],TotEvent[i]);
}









float PlayHand(card deal[], int weight)
{
        int i,DealInt[5],TotDiscard,total[15];
        float ev;
        for (i=0; i<=4; i++)
                DealInt[i]=4*deal[i].r+deal[i].s;
        ev=draw(DealInt,total,&TotDiscard);
        for (i=0; i<=14; i++)
                TotEvent[i]+=weight*DrawWeight[TotDiscard]*total[i];
```

```
	if (details==1)
	{
		for (int i=0; i<=4; i++)
			cerr << charrank[deal[i].r] << "/" <<
charsuit[deal[i].s] << "\t";
		cerr << "\nTotal Discards=\t" << TotDiscard << "\n";
		for (i=0; i<=14; i++)

	printf("%i\t%i\t%i\t%I64i\n",i,pay[i],total[i],TotEvent[i]);
		cerr << "ev=\t" << ev << "\n";
		cin >> i;
	}
	return ev;
}




void SetArray(void)
{
	int i,j,k,l,m,pt,r,s;
	card t[5],deck[52];
	pt=0;
	for (r=0; r<=12; r++)
	{
		for (s=0; s<=3; s++)
		{
			deck[pt].r=r;
			deck[pt].s=s;
//			deck[pt].in=1;
			pt++;
		}
	}
	pt=0;
	for (i=0; i<=47; i++)
	{
		t[0]=deck[i];
		for (j=i+1; j<=48; j++)
		{
			t[1]=deck[j];
			for (k=j+1; k<=49; k++)
			{
				t[2]=deck[k];
				for (l=k+1; l<=50; l++)
				{
					t[3]=deck[l];
					for (m=l+1; m<=51; m++)
					{
						t[4]=deck[m];
						ScoreArray[pt]=score(t);
						pt++;
					}
				}
			}
		}
```

```
        }
}


void sort(card d[]) // lowest to highest
{
      int i,j;
      card hold;
      for (i=3; i>=0; i--)
            for (j=0; j<=i; j++)
                  if (d[j].r > d[j+1].r)
                  {
                        hold=d[j];
                        d[j]=d[j+1];
                        d[j+1]=hold;
                  }
}



int score(card t[])
{
      int straight=0;
      int flush=0;
      if (t[0].r==t[3].r)
      {
            if (t[0].r==12)
                  return 9;
            else if ((t[0].r>=0)&&(t[0].r<=2))
            {
                  if ((t[4].r<=2)||(t[4].r==12))
                        return 10;
                  else
                        return 8;
            }
            else
                  return 7;
      }
      else if (t[1].r==t[4].r)
      {
            if (t[4].r==12)
            {
                  if (t[0].r<=2)
                        return 11;
                  else
                        return 9;
            }
            else if ((t[4].r>=0)&&(t[4].r<=2))
            {
                  if ((t[0].r<=2)||(t[0].r==12))
                        return 10;
                  else
                        return 8;
            }
            else
                  return 7;
      }
```

```
      else if ((t[0].r==t[1].r)&&(t[3].r==t[4].r) &&
      ((t[1].r==t[2].r)||(t[2].r==t[3].r)))
            return 6; // full house
      else if ((t[0].r==t[2].r)||(t[1].r==t[3].r)||(t[2].r==t[4].r))
            return 3; // three of a kind
      else if (((t[0].r==t[1].r)&&(t[2].r==t[3].r)) ||
      ((t[0].r==t[1].r)&&(t[3].r==t[4].r)) ||
      ((t[1].r==t[2].r)&&(t[3].r==t[4].r)))
            return 2; // two pair
      else if (((t[0].r==t[1].r)&&(t[0].r>=9))||
            ((t[1].r==t[2].r)&&(t[1].r>=9))||
            ((t[2].r==t[3].r)&&(t[2].r>=9))||
            ((t[3].r==t[4].r)&&(t[3].r>=9)))
            return 1; // pair
      else
      {
            if
((t[0].s==t[1].s)&&(t[1].s==t[2].s)&&(t[2].s==t[3].s)&&(t[3].s==t[4].s)
)
                  flush=1;
            if (((t[4].r==(t[0].r+4)) || ((t[3].r==3)&&(t[4].r==12)))
                  &&
(t[0].r!=t[1].r)&&(t[2].r!=t[1].r)&&(t[2].r!=t[3].r)&&(t[3].r!=t[4].r))
                  straight=1;
            if ((flush==1)&&(straight==1))
            {
                  if (t[0].r==8)
                        return 13; // royal flush
                  else
                        return 12; // straight flush
            }
            else if (flush==1)
                  return 5;
            else if (straight==1)
                  return 4;
            else
                  return 0;
      }
}



int FourToRoyalTest(card t[])
{
      if
((t[0].r>=8)&&(t[0].s==t[1].s)&&(t[1].s==t[2].s)&&(t[2].s==t[3].s))
            return 1;
      else if
((t[1].r>=8)&&(t[3].s==t[4].s)&&(t[1].s==t[2].s)&&(t[2].s==t[3].s))
            return 1;
      else
            return 0;
}
```

```
void FourToRoyalEvent(int c0, int c1, int c2, int c3, int c4, int
deal[], int Four2RoyalEvent[])
{
      int i,j,pt,suit;
      card draw1[5],draw2[5],temp[5];
      draw1[0].r=c0/4;
      draw1[0].s=c0%4;
      draw1[1].r=c1/4;
      draw1[1].s=c1%4;
      draw1[2].r=c2/4;
      draw1[2].s=c2%4;
      draw1[3].r=c3/4;
      draw1[3].s=c3%4;
      draw1[4].r=c4/4;
      draw1[4].s=c4%4;
      if (draw1[0].s==draw1[1].s)
            suit=draw1[0].s;
      else
            suit=draw1[2].s;
      pt=0;
      for (i=0; i<=4; i++)
      {
            if ((draw1[i].s==suit)&&(draw1[i].r>=8))
            {
                  temp[pt].r=draw1[i].r;
                  temp[pt].s=draw1[i].s;
                  pt++;
            }
      }
      for (i=0; i<=14; i++)
            Four2RoyalEvent[i]=0;
      for (i=0; i<=51; i++)
      {
            if
((i!=deal[0])&&(i!=deal[1])&&(i!=deal[2])&&(i!=deal[3])&&(i!=deal[4])&&
(i!=c0)&&(i!=c1)&&(i!=c2)&&(i!=c3)&&(i!=c4))
            {
                  for (j=0; j<=3; j++)
                        draw2[i]=temp[i];
                  draw2[4].r=i/4;
                  draw2[4].s=i%4;
                  sort(draw2);
                  Four2RoyalEvent[score(draw2)]++;
            }
      }
}




void entercards(card deal[],int numcards)
{
      int i,j;
      char entcard[2];
```

```
      char
charrank[20]={'2','3','4','5','6','7','8','9','t','j','q','k',
        'a','T','J','Q','K','A','w','W'};
      char charsuit[10]={'h','c','d','s','H','C','D','S','w','W'};
      int
intrank[20]={0,1,2,3,4,5,6,7,8,9,10,11,12,8,9,10,11,12,13,13};
      int intsuit[10]={0,1,2,3,0,1,2,3,0,0};
      for (i=0; i<numcards; i++)
      {
            cerr << "Enter your card #" << i << ":  ";
            cin >> entcard;
            for (j=0; j<=19; j++)
                  if (entcard[0]==charrank[j])
                        deal[i].r=intrank[j];
            for (j=0; j<=9; j++)
                  if (entcard[1]==charsuit[j])
                        deal[i].s=intsuit[j];
      }
}



void presetup1()
{
      int i,j,k,l,m;
      card player[5];

      // Set all hands with no matching rank (65637 total)
      for (i=0; i<=8; i++)
      {
            for (j=i+1; j<=9; j++)
            {
                  for (k=j+1; k<=10; k++)
                  {
                        for (l=k+1; l<=11; l++)
                        {
                              for (m=l+1; m<=12; m++)
                              {
                                    // all suits equal
                                    setrank(i,j,k,l,m,player);

                                    setsuit(1,1,1,1,1,player,4);

                                    // all suits equal except one card
                                    setsuit(2,1,1,1,1,player,12);
                                    setsuit(1,2,1,1,1,player,12);
                                    setsuit(1,1,2,1,1,player,12);
                                    setsuit(1,1,1,2,1,player,12);
                                    setsuit(1,1,1,1,2,player,12);

                                    // three cards one suit, two cards
another
                                    setsuit(2,2,1,1,1,player,12);

                                    setsuit(2,1,2,1,1,player,12);
```

```
setsuit(2,1,1,2,1,player,12);
setsuit(2,1,1,1,2,player,12);
setsuit(1,2,2,1,1,player,12);
setsuit(1,2,1,2,1,player,12);
setsuit(1,2,1,1,2,player,12);
setsuit(1,1,2,2,1,player,12);
setsuit(1,1,2,1,2,player,12);
setsuit(1,1,1,2,2,player,12);
// three cards one suit, fourth and
fifth two other ranks
setsuit(2,3,1,1,1,player,24);
setsuit(2,1,3,1,1,player,24);
setsuit(2,1,1,3,1,player,24);
setsuit(2,1,1,1,3,player,24);
setsuit(1,2,3,1,1,player,24);
setsuit(1,2,1,3,1,player,24);
setsuit(1,2,1,1,3,player,24);
setsuit(1,1,2,3,1,player,24);
setsuit(1,1,2,1,3,player,24);
setsuit(1,1,1,2,3,player,24);
// two cards one suit, two cards
another, fifth card a third
setsuit(1,1,2,2,3,player,24);
setsuit(1,2,1,2,3,player,24);
setsuit(1,2,2,1,3,player,24);
setsuit(1,1,2,3,2,player,24);
setsuit(1,2,1,3,2,player,24);
setsuit(1,2,2,3,1,player,24);
setsuit(1,1,3,2,2,player,24);
setsuit(1,2,3,1,2,player,24);
```

```
                                        setsuit(1,2,3,2,1,player,24);
                                        setsuit(1,3,1,2,2,player,24);
                                        setsuit(1,3,2,1,2,player,24);
                                        setsuit(1,3,2,2,1,player,24);
                                        setsuit(3,1,1,2,2,player,24);
                                        setsuit(3,1,2,1,2,player,24);
                                        setsuit(3,1,2,2,1,player,24);
                                        // two cards one suit, all other
suits have one card each
                                        setsuit(4,4,1,2,3,player,24);
                                        setsuit(4,1,4,2,3,player,24);
                                        setsuit(4,2,3,4,1,player,24);
                                        setsuit(4,1,2,3,4,player,24);
                                        setsuit(1,4,4,2,3,player,24);
                                        setsuit(1,4,2,4,3,player,24);
                                        setsuit(1,4,2,3,4,player,24);
                                        setsuit(2,3,4,4,1,player,24);
                                        setsuit(2,3,4,1,4,player,24);
                                        setsuit(1,2,3,4,4,player,24);
                                    }
                                }
                            }
                        }
                    }
                }

void presetup2()
{
        int i,j,k,l;
        card player[5];

        // Pairs (91520 total)
        for (i=0; i<=9; i++)
        {
                for (j=i+1; j<=10; j++)
                {
                        for (k=j+1; k<=11; k++)
                        {
                                for (l=k+1; l<=12; l++)
                                {
```

```
setrank(i,i,j,k,l,player);

setsuit(1,2,1,1,1,player,12);
setsuit(1,2,1,1,2,player,12);
setsuit(1,2,1,2,1,player,12);
setsuit(1,2,2,1,1,player,12);
setsuit(1,2,1,1,3,player,24);
setsuit(1,2,1,3,1,player,24);
setsuit(1,2,3,1,1,player,24);
setsuit(1,2,1,3,3,player,24);
setsuit(1,2,3,1,3,player,24);
setsuit(1,2,3,3,1,player,24);
setsuit(1,2,3,3,3,player,12);
setsuit(1,2,1,2,3,player,24);
setsuit(1,2,1,3,2,player,24);
setsuit(1,2,3,1,2,player,24);
setsuit(1,2,3,4,4,player,24);
setsuit(1,2,4,3,4,player,12);
setsuit(1,2,4,4,3,player,12);
setsuit(1,2,1,3,4,player,12);
setsuit(1,2,3,1,4,player,24);
setsuit(1,2,3,4,1,player,24);

setrank(i,j,j,k,l,player);

setsuit(1,1,2,1,1,player,12);
setsuit(1,1,2,1,2,player,12);
setsuit(1,1,2,2,1,player,12);
setsuit(2,1,2,1,1,player,12);
setsuit(1,1,2,1,3,player,24);
setsuit(1,1,2,3,1,player,24);
setsuit(3,1,2,1,1,player,24);
setsuit(1,1,2,3,3,player,24);
setsuit(3,1,2,1,3,player,24);
setsuit(3,1,2,3,1,player,24);
setsuit(3,1,2,3,3,player,12);
setsuit(1,1,2,2,3,player,24);
setsuit(1,1,2,3,2,player,24);
setsuit(3,1,2,1,2,player,24);
setsuit(3,1,2,4,4,player,24);
setsuit(4,1,2,3,4,player,12);
setsuit(4,1,2,4,3,player,12);
setsuit(1,1,2,3,4,player,12);
setsuit(3,1,2,1,4,player,24);
setsuit(3,1,2,4,1,player,24);


setrank(i,j,k,k,l,player);

setsuit(1,1,1,2,1,player,12);
setsuit(1,1,1,2,2,player,12);
setsuit(1,2,1,2,1,player,12);
setsuit(2,1,1,2,1,player,12);
setsuit(1,1,1,2,3,player,24);
setsuit(1,3,1,2,1,player,24);
setsuit(3,1,1,2,1,player,24);
setsuit(1,3,1,2,3,player,24);
```

```
                                    setsuit(3,1,1,2,3,player,24);
                                    setsuit(3,3,1,2,1,player,24);
                                    setsuit(3,3,1,2,3,player,12);
                                    setsuit(1,2,1,2,3,player,24);
                                    setsuit(1,3,1,2,2,player,24);
                                    setsuit(3,1,1,2,2,player,24);
                                    setsuit(3,4,1,2,4,player,24);
                                    setsuit(4,3,1,2,4,player,12);
                                    setsuit(4,4,1,2,3,player,12);
                                    setsuit(1,3,1,2,4,player,12);
                                    setsuit(3,1,1,2,4,player,24);
                                    setsuit(3,4,1,2,1,player,24);


                                    setrank(i,j,k,l,l,player);

                                    setsuit(1,1,1,1,2,player,12);
                                    setsuit(1,1,2,1,2,player,12);
                                    setsuit(1,2,1,1,2,player,12);
                                    setsuit(2,1,1,1,2,player,12);
                                    setsuit(1,1,3,1,2,player,24);
                                    setsuit(1,3,1,1,2,player,24);
                                    setsuit(3,1,1,1,2,player,24);
                                    setsuit(1,3,3,1,2,player,24);
                                    setsuit(3,1,3,1,2,player,24);
                                    setsuit(3,3,1,1,2,player,24);
                                    setsuit(3,3,3,1,2,player,12);
                                    setsuit(1,2,3,1,2,player,24);
                                    setsuit(1,3,2,1,2,player,24);
                                    setsuit(3,1,2,1,2,player,24);
                                    setsuit(3,4,4,1,2,player,24);
                                    setsuit(4,3,4,1,2,player,12);
                                    setsuit(4,4,3,1,2,player,12);
                                    setsuit(1,3,4,1,2,player,12);
                                    setsuit(3,1,4,1,2,player,24);
                                    setsuit(3,4,1,1,2,player,24);


                              }
                        }
                  }
            }
      }


void presetup3()
{
      int i,j,k;
      card player[5];

      // Set all two pair hands (6864 total)
      for (i=0; i<=10; i++)
      {
            for (j=i+1; j<=11; j++)
            {
                  for (k=j+1; k<=12; k++)
                  {
```

```
                        setrank(i,i,j,j,k,player);
                        setsuit(1,2,3,4,1,player,12);
                        setsuit(1,2,3,4,3,player,12);
                        setsuit(1,2,1,3,1,player,24);
                        setsuit(1,2,1,3,2,player,24);
                        setsuit(1,2,1,3,3,player,24);
                        setsuit(1,2,1,3,4,player,24);
                        setsuit(1,2,1,2,1,player,12);
                        setsuit(1,2,1,2,3,player,12);

                        setrank(i,i,j,k,k,player);
                        setsuit(1,2,1,3,4,player,12);
                        setsuit(1,2,3,3,4,player,12);
                        setsuit(1,2,1,1,3,player,24);
                        setsuit(1,2,2,1,3,player,24);
                        setsuit(1,2,3,1,3,player,24);
                        setsuit(1,2,4,1,3,player,24);
                        setsuit(1,2,1,1,2,player,12);
                        setsuit(1,2,3,1,2,player,12);

                        setrank(i,j,j,k,k,player);
                        setsuit(1,1,2,3,4,player,12);
                        setsuit(3,1,2,3,4,player,12);
                        setsuit(1,1,2,1,3,player,24);
                        setsuit(2,1,2,1,3,player,24);
                        setsuit(3,1,2,1,3,player,24);
                        setsuit(4,1,2,1,3,player,24);
                        setsuit(1,1,2,1,2,player,12);
                        setsuit(3,1,2,1,2,player,12);
                    }
                }
            }
        }


void presetup4()
{
        int i,j,k;
        card player[5];

        // Set all three of a kind hands (4290 total)
        for (i=0; i<=10; i++)
        {
                for (j=i+1; j<=11; j++)
                {
                        for (k=j+1; k<=12; k++)
                        {
                                setrank(i,i,i,j,k,player);
                                setsuit(1,2,3,1,2,player,24);
                                setsuit(1,2,3,1,4,player,12);
                                setsuit(1,2,3,4,1,player,12);
                                setsuit(1,2,3,1,1,player,12);
                                setsuit(1,2,3,4,4,player,4);

                                setrank(i,j,j,j,k,player);
                                setsuit(1,1,2,3,2,player,24);
```

```
                        setsuit(1,1,2,3,4,player,12);
                        setsuit(4,1,2,3,1,player,12);
                        setsuit(1,1,2,3,1,player,12);
                        setsuit(4,1,2,3,4,player,4);

                        setrank(i,j,k,k,k,player);
                        setsuit(1,2,1,2,3,player,24);
                        setsuit(1,4,1,2,3,player,12);
                        setsuit(4,1,1,2,3,player,12);
                        setsuit(1,1,1,2,3,player,12);
                        setsuit(4,4,1,2,3,player,4);

                }
            }
        }
}


void presetup5()
{
        int i,j;
        card player[5];

        // Set all full house hands (312 total)
        for (i=0; i<=11; i++)
        {
            for (j=i+1; j<=12; j++)
            {
                setrank(i,i,j,j,j,player);
                setsuit(1,2,1,2,3,player,12);
                setsuit(1,4,1,2,3,player,12);

                setrank(i,i,i,j,j,player);
                setsuit(1,2,3,1,2,player,12);
                setsuit(1,2,3,1,4,player,12);
            }
        }
}


void presetup6() // Set all four of a kind hands (156 total)
{
        int i,j;
        card player[5];

        for (i=0; i<=11; i++)
        {
            for (j=i+1; j<=12; j++)
            {

                setrank(i,i,i,i,j,player);
                setsuit(1,2,3,4,1,player,4);

                setrank(i,j,j,j,j,player);
                setsuit(1,1,2,3,4,player,4);
```

```
            }
        }
}



void setsuit(int a, int b, int c, int d, int e, card player[], int w)
{
      // Total classes of hands = 134459
      numhand++;
      TotalWeight+=w;
      player[0].s=a-1;
      player[1].s=b-1;
      player[2].s=c-1;
      player[3].s=d-1;
      player[4].s=e-1;
      sort(player);
      if (numhand%1==1)
      {
            details=1;
      }
      else
            details=0;
      PlayHand(player,w);

      curtime=time(NULL);
      if (numhand%50==0)
            cerr << "min rem. " << (float)(curtime-
begtime)*(float)(134459-numhand)/(float)numhand/60 << "\n";

}



void setrank(int a, int b, int c, int d, int e, card player[])
{
      player[0].r=a;
      player[1].r=b;
      player[2].r=c;
      player[3].r=d;
      player[4].r=e;
}



float draw(int deal[], int total[], int *TotDiscard)
{
      int i,j,k,l,m,pt,count,total2d[32][14],TotRet[32],TotCom[32];
      count=0;
      for (i=0; i<=14; i++)
            total[i]=0;
      for (i=0; i<=31; i++)
            for (j=0; j<=14; j++)
                  total2d[i][j]=0;
```

```
	for (i=0; i<=47; i++)
	{
		for (j=i+1; j<=48; j++)
		{
			for (k=j+1; k<=49; k++)
			{
				for (l=k+1; l<=50; l++)
				{
					for (m=l+1; m<=51; m++)
					{
						pt=0;
						if
((i==deal[0])||(j==deal[0])||(k==deal[0])||(l==deal[0])||(m==deal[0]))
							pt+=1;
						if
((i==deal[1])||(j==deal[1])||(k==deal[1])||(l==deal[1])||(m==deal[1]))
							pt+=2;
						if
((i==deal[2])||(j==deal[2])||(k==deal[2])||(l==deal[2])||(m==deal[2]))
							pt+=4;
						if
((i==deal[3])||(j==deal[3])||(k==deal[3])||(l==deal[3])||(m==deal[3]))
							pt+=8;
						if
((i==deal[4])||(j==deal[4])||(k==deal[4])||(l==deal[4])||(m==deal[4]))
							pt+=16;
						total2d[pt][ScoreArray[count]]++;
						count++;
					}
				}
			}
		}
	}
	for (i=0; i<=31; i++)
	{
		TotRet[i]=0;
		TotCom[i]=0;
	}
	for (pt=0; pt<=31; pt++)
	{
		for (i=0; i<=13; i++)
		{
			TotRet[pt]+=total2d[pt][i]*pay[i];
			TotCom[pt]+=total2d[pt][i];
		}
	}
	float ev=0;
	float ev_max=0;
	int pt_max=0;
	for (pt=0; pt<=31; pt++)
	{
		ev=(float)TotRet[pt]/(float)TotCom[pt];
		if (ev>ev_max)
		{
			ev_max=ev;
			pt_max=pt;
		}
```

```
        }
        for (i=0; i<=13; i++)
        {
              total[i]=total2d[pt_max][i];
              total[14]+=total2d[pt_max][i];
        }
        *TotDiscard=DiscardNum[pt_max];
        return ev_max;
}
```